CAP – Canonical Abstract Prototypes for Abstract Visual and Interaction Design

Larry L. Constantine
University of Technology, Sydney, (Australia)
Constantine & Lockwood Ltd, lconstantine@foruse.com

Canonical Abstract Prototypes

- **Prototype**
  - Artifact that embodies design ideas, solutions and decisions without being the actual user interface
  - Often classified according to
    - concreteness or fidelity, i.e. how much it looks like the final result
    - how functional or deep it is, i.e. how real it's behaviour is
    - how long it is meant to last (through away or evolve into real)

- **Abstract prototype**
  - Prototype that is (a lot) less concrete than a high-fidelity prototype
  - An abstract prototype concentrates on structure and behavior and avoids focussing on look & feel

- **Canonical abstract prototype**
  - A prototype that is based on a standard (i.e. canonical) set of interaction elements
CAP

- Lightweight notation
  - containment hierarchy, i.e. box inside box
  - classification of interaction elements
  - no formal semantics
  - diagram layout approximates screen layout

From abstract:
"...intermediate between abstract task models and realistic or representational prototypes."

"...provides a formal vocabulary for expressing visual and interaction designs without concern for details of appearance and behavior."

How would you classify CAP?

formality

perspective

granularity

CAP scope
Usage Centered Design – UsageCD

- **CAP** is part of a method named **Usage Centered Design**
  - should not be confused with **User Centered Design**, shortened **UCD**
  - based on models of roles and tasks
  - task model uses a certain style of use cases, named **essential use case**

- **Bevare**
  - UsageCD has been accused of (re)inventing terms
  - UsageCD is less novel and innovative than the impression they (want to) give
  - few papers and a lot of self-referencing

- **Nevertheless: a practical and light-weight method**

---

Essential Use Case

  "[...] is a simplified, abstract, generalized use case that captures the intentions of a user in a technology and implementation independent manner"

- describes a specific task that the user needs to perform, without unnecessary clutter and detail, like stories and prose or mode of interaction with a specific technology

- the form is deliberately terse and compact

- an essential use case will normally correspond to a specific and small part of the user interface, partly because it is focussed, partly because that's the way the method works
Content modelling

- model of "what the user interface contains"
  - defines the necessary functionality
  - partitions the interface into logical elements
- bridges the gap between task models and concrete design representations
- a continuum of abstractions
  - lists of information, functions and interface elements spread across interaction units, like frames, windows and panes and dialogs
  - wire-frame diagrams indicate relative importance with position (left-most elements are read first) and size (biggest elements are easiest to discover)
  - sketches, paper prototypes, high-fidelity prototypes, accurate mockups and vertical prototypes
Canonical Abstract Prototypes

- bridge between task models and concrete design representations

"[...] a model specifically created to support a smooth progression from abstraction toward realization in user interface design."

Task models  Abstract prototypes  Concrete prototypes with canonical components

- "[...] are constructed from a standardized set of universal abstract components"

Canonical Abstract Component

- standardized user interface elements

- one symbol for each standard interactive function

  - action, material, active material

- symbols are combined and elaborated to represent more specific functions

  - collection, selection and selectable collection

- symbols are designed to be intuitive and easy to recognize (as opposed to recall)
Standard actions

- list of actions typical of desktop applications
- can be modelled as `<verb> <noun>` phrase
- correspond to interactive function, not UI element

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>INTERACTIVE FUNCTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>action/operation*</td>
<td>Print symbol table, Color selected shape</td>
</tr>
<tr>
<td>🔄</td>
<td>start/go/to</td>
<td>Begin consistency check, Confirm purchase</td>
</tr>
<tr>
<td>🔄</td>
<td>stop/end/complete</td>
<td>Finish inspection session, Interrupt test</td>
</tr>
<tr>
<td>✕</td>
<td>select</td>
<td>Group member picker, Object selector</td>
</tr>
<tr>
<td>✕</td>
<td>create</td>
<td>New customer, Blank slide</td>
</tr>
<tr>
<td>✕</td>
<td>delete, erase</td>
<td>Break connection line, Clear form</td>
</tr>
<tr>
<td>✕</td>
<td>modify</td>
<td>Change shipping address, Edit client details</td>
</tr>
<tr>
<td>✕</td>
<td>move</td>
<td>Put into address list, Move up/down</td>
</tr>
<tr>
<td>✕</td>
<td>duplicate</td>
<td>Copy address, Duplicate slide</td>
</tr>
<tr>
<td>✕</td>
<td>perform (&amp; return)</td>
<td>Object formatting, Set print layout</td>
</tr>
<tr>
<td>✕</td>
<td>toggle</td>
<td>Bold on/off, Encrypted mode</td>
</tr>
<tr>
<td>✕</td>
<td>view</td>
<td>Show file details, Switch to summary</td>
</tr>
</tbody>
</table>

Standard presentation components

- Abstract material seems to correspond to presentation components, i.e. components focused on presenting information or objects

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>INTERACTIVE FUNCTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>🔄</td>
<td>container*</td>
<td>Configuration holder, Employee history</td>
</tr>
<tr>
<td>🔄</td>
<td>element</td>
<td>Customer ID, Product thumbnail image</td>
</tr>
<tr>
<td>🔄</td>
<td>collection</td>
<td>Personal addresses, Electrical Components</td>
</tr>
<tr>
<td>🔄</td>
<td>notification</td>
<td>Email delivery failure, Controller status</td>
</tr>
</tbody>
</table>
Hybrid components

- Hybrid components combine information presentation (output) and user input

### Table: Hybrid Components

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>INTERACTIVE FUNCTION</th>
<th>EXAMPLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>📇 active material*</td>
<td>Expandable thumbnail, Resizable chart</td>
<td></td>
</tr>
<tr>
<td>📇 input/accepter</td>
<td>Accept search terms, User name entry</td>
<td></td>
</tr>
<tr>
<td>📇 editable element</td>
<td>Patient name, Next appointment date</td>
<td></td>
</tr>
<tr>
<td>📇 editable collection</td>
<td>Patient details, Text object properties</td>
<td></td>
</tr>
<tr>
<td>📇 selectable collection</td>
<td>Performance choices, Font selection</td>
<td></td>
</tr>
<tr>
<td>📇 selectable action set</td>
<td>Go to page, Zoom scale selection</td>
<td></td>
</tr>
<tr>
<td>📇 selectable view set</td>
<td>Choose patient document, Set display mode</td>
<td></td>
</tr>
</tbody>
</table>

- Since these concepts are not formally defined, it is not always clear which one to use, e.g. what is the difference between collection and view set?

Design process – from task model to CAP

- a systematic process, not a mechanical one
- related tasks/use cases are clustered or grouped into an interaction context
  - "related" means that tasks use related information and are relevant to perform at the same time
  - an interaction context typically corresponds to a window, pane or dialog and is the scope of a diagram
- the interaction context is step by step populated by the material and tools needed for performing the tasks that are grouped into it
  - if a task needs some information, add material
  - if a task needs function, add tool
  - if a task edits information, add active material
Design process – from task model to CAP

- the components in an interaction context are laid out according to domain constraints
  - area is allocated based on importance, relevance, user focus, complexity etc.
  - position is determined based on workflow (task sequence), semantic relations, etc.
  - grouping of components are used to aid the user in understanding the relation between components

- concrete interaction elements (widgets) are selected according to the abstract components’ interactive function
  - for each component there are usually few alternatives
  - rules for mapping from abstract to concrete exist

From abstract to concrete components

- Abstract components are mapped to concrete
  - usually few alternatives exist, choice may be obvious
  - some concrete components may support several functions, e.g., both view and selection
  - some rearrangement may be necessary/desirable
  - may depend on choice of platform and/or device
Abstract design patterns

- A design pattern suggests a generic (and hence abstract) solution for a reoccurring problem.
- Usually semi-structured description of:
  - problem – when this pattern is applicable
  - issues or “forces” – important aspects of the problem affecting and constraining the solution
  - solution – how the issues are resolved and components put together to form a generic solution
  - examples indicating how to apply the pattern and adapt it to the concrete problem

Detail View Navigation pattern

- problem – that of exploring list of elements and their details
- issues – getting overview and enough information before exploring individual elements

- solution
  - list view with columns
  - popup detail window with navigation tools
Summary of Canonical Abstract Prototypes

- part of the Usage Centered Design method, fills the gap between task models and concrete design
- based on role and task models
- abstract prototype is composed of standardized interaction components, based on desired interactive function
- abstract components are subsequently mapped to concrete interaction elements
- design problem solutions may reused by means of so-called abstract design patterns, that embody generic solutions to reoccurring problems