Notes - Design - Information Architecture

- Dr Nick Hayward

A brief consideration of useful hints and tips for an application's information architecture.

Contents

Intro Information architecture is concerned with the organisation of information into a perceived coherent structure. This structure is considered comprehensive, navigable, and in many situations searchable. For example, concepts, entities, relationships, functionality, events, content, and so on.

When designing such information architecture, we often need to consider and implement the following,

- a data model
- a naming scheme or glossary
- names and titles for identification of places
- navigation and location awareness
- navigation map and associated mechanisms
- breadcrumbs and navigation notifications
- presentation of such places
- searching

Data model, naming scheme, naming places... Development of a data model for an application includes the identification and recording of the entities, attributes, and operations for each entity. It also includes identification of the relationships between the entities.

It is often argued that the data model is, in fact, part of the app's interaction concept. This is because it is perceived to help define the nature of the product, and because its entities, attributes, and operations will have an impact on the visual design itself, and its associated behaviour.

A coherent and consistent naming scheme is important as it helps users form a correct mental model of the functionality of an application. Definition of official names for key elements and processes in an application can also be formalised and recorded in the defined interaction concept. For some applications, in particular those with complex, specialised domains, a glossary of names and labels is a particularly useful concept. It helps define the official, preferred terminology for the entire application. With this in place, the interaction concept may simply link or reference this glossary.

Naming of places is equally useful and important. Each place within our application should be clearly named and labelled, obviously to help our uses determine what they are looking at and where they are in the application. Naturally, this also helps our users easily differentiate places and concepts within the application.

Such naming of places also allows us to easily define them within menus, instructions, help text, search options, and so on. Also, don't forget that whilst official, pre-defined names may pre-dominate within your application, it is also OK for a place to adopt a user-defined name. This is normally the case for an editing application or user centric information.

Navigation and places For application design, we often reference navigation relative to defined places. For example, in a web application places may be defined as pages or screens. Not all of these places need necessarily be user accessible, but they are still defined as places. Places may also refer to sub-divisions such as panels, tabs, and sub-sections of a given screen as well. These sub-sections may include dialogs, image presentations, and so on.

For a design with many such examples of places, a design should help users determine and differentiate,

- where they are currently located within the app
- where they can go next
- how to easily get where they want to go

As defined, one aspect of this navigation is the concept of identification of places with names and titles. However, we must also consider the actual presentation of places within the application. For example,

- how do we present different places to our users? For example, are they all defined equally or do we use highlighting and focusing to draw our users' attention.
- will a user be able to view multiple places at once, or must they page or navigate their way through single places? For example, consider the difference between a desktop website and a responsive, mobile rendering. We can also see this type of basic consideration for navigation controllers in iOS design.
- can these places be resized, moved and rearranged, opened, closed, hidden, removed entirely, and so on.
- can we relate content from one place to another. If yes, how do we present and relay this related, contextual information to our users within each respective place. If the data can be edited in one place, does it affect and modify the data in the related place? These considerations will also affect and impinge upon your interaction concepts, and the underlying information architecture.

Navigation map Navigation maps, from a design and development perspective, allow us to consider and define the places that may exist within our application, and the movements allowed from one to the other. These navigation maps are often most beneficial if represented in a graphical manner within quick reference diagrams.

Navigation maps are not always relevant for all applications, and there will be occasions where designing a complete navigation map at the design stage is both impractical and counter-productive. An initial map can always be expanded and modified as we develop the application.

There will also be instances where a navigation map is simply impractical. For example, if we consider certain dynamic applications, such as catalogues and wikis, we can start to see how many different links, pathways, and related material a user may generate.

Navigation mechanisms There are many different ways for a user to switch places and content. However, there are some standard defined examples such as

- menus
- links
- buttons
- hierarchical structures - eg: trees may be used to represent the hierarchical depth of the displayed data, document etc. So, it might be beneficial to display the hierarchical outline of the data in one pane relative to the current document or data
- maps - an application may present data points etc relative to geographical locations, or simply design a conceptual map of the application or domain itself for reference
- flow diagrams - allow a user to visualise steps and outcomes relative to the current complex process or workflow. For example, we might visually represent the flow for a calculation or time-sensitive process.
- switching - allows a user to easily move between multiple places that are currently available within the UI
- events - an event triggered by a user action or application process should also show a notification or message window. Some sort of temporary pop-up might be shown to quickly alert the user…
- searching - the simple act of searching by keyword, for example, or selecting from a faceted list of terms is another form of user navigation
- history - some applications present chronological lists of recently viewed or requested information and places. Web browsers, of course, are a good example of this type of navigation option.
- bookmarks / favourites - content driven and focused applications will often allow users to bookmark content and places to allow for quick, easy recall.

User location As the scale and complexity of an app increases, it becomes inherently more important to clearly identify a user's current location. It acts as a quick reminder to the user, and also creates a familiar contextual placeholder within the application.

We can indicate the user's current location in a number of different ways. For example,

- clearly display the title or name of the current place with any associated contextual name. This might include the place name and the document name or title, for example.
- highlight the current place name or title on a visual map or flow diagram. This could also include a representation of location on a visual flow diagram for a process or series of tasks.
- we can also locate a current place within a defined hierarchical structure, such as a tree representation of the current document or data…

For many applications with hierarchical data representations, we can also add the common **breadcrumb** trail and indicator to the current place. A **breadcrumb** has the benefit of acting as both an indication of the user's current location and as a simple form of navigation.

Considerations For our applications, we can identify core sets of features, tasks, actions, operations, and processes. We can also consider series of use cases that follow and share similar patterns of interaction.

For example, an editing application may allow user interaction with many disparate tools and actions via a common menu structure. A user selects a given tool, which then allows data entry or manipulation. The variance is the selected tool itself, but the interaction will be able to follow a similar pattern. We can also see this with games, where many different levels, challenges, and opponents can be targeted by our user using similar interaction concepts from level to level.

For such applications, and interaction scenarios, it makes sense to create an initial list or breakdown of these similar tasks or features. We can then start to design an interaction framework to describe perceived commonalities in the presentation and behaviour of the user interface.

The creation of this list and overview allows us, as designers and developers, to understand how our application will fundamentally behave. It also helps ensure consistency across such similar tasks, thereby allowing our users to develop correct mental models.

The other benefit is that by simply documenting the commonalities between such tasks, it will save us from re-documenting the same aspects for individual tasks as we compile and write our overall specifications.

This framework will also be useful for the development of the overall design, and the technical underpinnings of the application itself.

Issues So, some of the issues you might need to consider for your interaction framework are as follows,

- how the tasks are started or triggered, such as a user selecting an item on a menu
- any required authorisations (ie: which tasks can be started and completed by which group of users…)
- when and how tasks can be activated, and any given cases where tasks may be disabled
- how and when the task is considered complete
- does the start or end of a task signal a change in any status, mode etc…
- what are the effects of the task on the system's data
  - eg: is the data saved automatically, does it persist or is it temporary, what happens if the task is abandoned or an error breaks the task, and so on…

Considering and designing the framework for interactions allows us to ensure that we understand, as developers and designers, how the application will fundamentally behave. It helps us try to ensure consistency across tasks, so that our users can perceive patterns and, thereby, once more form correct mental models.

Data and persistency A consideration of persistence and transactions in an application may also be considered relative to the interface design. We would normally consider what, if any, of the application's data needs to be stored in a persistent nature.

Relative to the interface and interaction concepts, we need to consider how the actual saving of data works in the application. For example, is the data generated by user interactions saved in a persistent store, or is it saved in a temporary cache for quicker read, write access.

We also need to consider how such data saving and persistency is relayed to the user. Are they fully aware that the data is being saved? Is it an explicit act in the interface design? eg: the user has to actually press a save button or menu item. Or, is it part of an auto-save option running as a background process.

As we consider a plan and outline for data storage and persistency, we often consider many disparate concepts. We will, naturally, consider standard data design patterns that include required validations of the data, and any accompanying error messages. Relative to the interaction and interface designs, we would need to carefully plan how our error messages are presented and, of course, whether the validation occurs on the client or server side. Your chosen application environment, whether it be mobile, web, desktop etc, will influence this choice greatly.

We would also need to consider whether partial data can be saved for incomplete interface tasks, such as a user completing only part of a form. In such an example, it might be useful to temporarily save the partial form data to allow a user to quickly and easily return to complete the form or document. It might also allow you to offer quick suggestions for previous edits for such material in future tasks.

Relative to the interface design, your chosen save points in the flow of a task will also impact notification points and any client-side to backend calls. For example, if you do not clearly define points in a process where data may be saved or cached, you may not correctly inform the user of save events, suggestions, any time limits, percentage left to complete, and so on. Such save points also allow us to easily keep track of whether data is currently saved or unsaved, assuming you have not implemented an auto-save feature.

There is a lot more to consider, but this is really better suited for a systems architecture, databases etc course.

References

- Norman, D. *The Design of Everyday Things.* Basic Books. 2013.