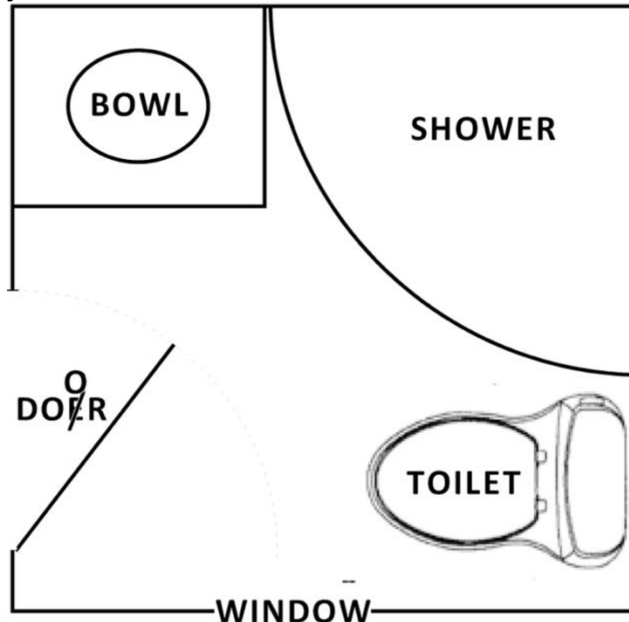
The DOer, SHOWer Pattern

Why the funny spelling? So you won't confuse the names with a misspelled architectural drawing like this one, and you will be helped to remember the intended pronunciation:

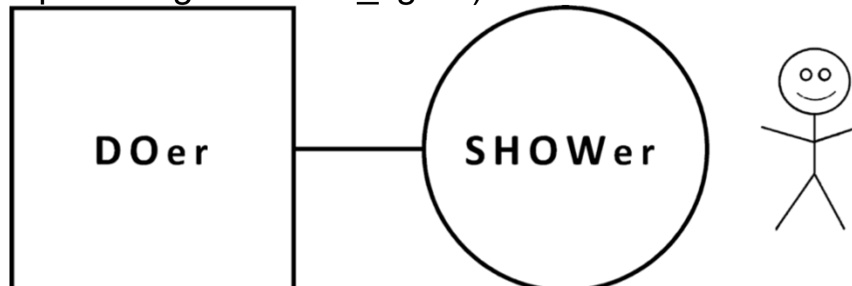
DOer, SHOWer rhymes with MOO-er, BLOW-er.



1/39

The pattern is simple. It has two parts: one thing does, one thing shows. It is a generalization of the Model-View-Controller pattern. Here is a figure we will be using.

(Stick figure courtesy of Wikipedia, created by Jleedev: https://en.wikipedia.org/wiki/Stick_figure)



2/39

The pattern is widely applicable in many domains. I invite you to add to it. It has two parts. As you will see later they are not necessarily software objects.
-The DOer is a thing that has a specific job that it Does.
3/39

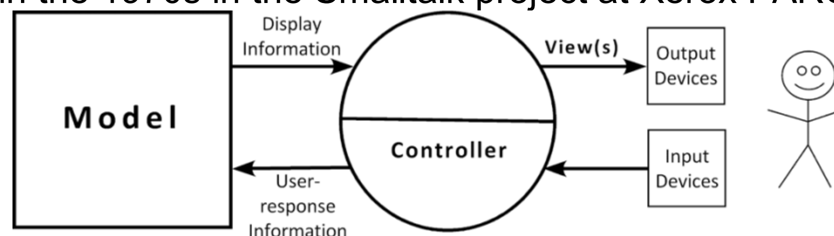
-The SHOWer is a user interface on the DOer. It permits the user (shown as a stick figure on the right) to see aspects of the DOer that the SHOWer's designer wants the user to see, and for the user to change or edit (some of) those aspects.
4/39

The SHOWer might be part-time. For example, it might exist as a factory that creates a dialog box object when the user requests.

The user is not part of the pattern.
5/39

Example 1: Model-View-Controller (MVC)

Model-View-Controller is the underlying model for the GUIs (graphical user interfaces) of almost all well-architected contemporary software. It was formalized in the 1970s in the Smalltalk project at Xerox PARC.



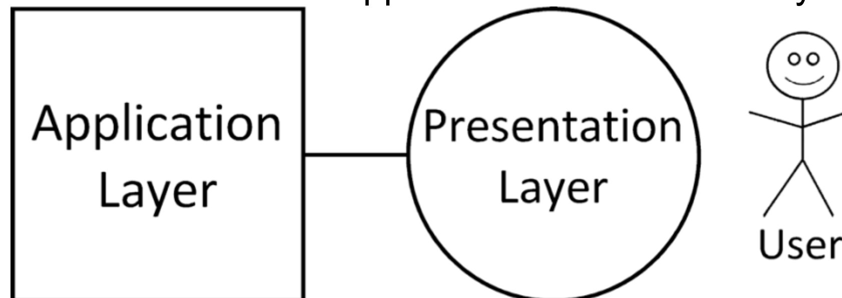
6/39

(In many descriptions, the terms View and Controller are not parallel; "View" means a display in a particular format (there might be several), and "Controller" is the part of the software that handles events from the user. Hence the asymmetry in the figure.)
7/39

©2020 Melvin Conway

Example 2: Application-Presentation Layered Model

The common GUI operating systems such as Smalltalk, Microsoft® Windows®, and Macintosh® OS X® employ variants of MVC. As a class of models we can refer to them as Application-Presentation Layered Models:



8/39

Example 3: The Application-Development Life Cycle (Conventional)

Here we see that the pattern is hierarchical, and we are prompted to think, not of the artifact, but of its life cycle.
9/39

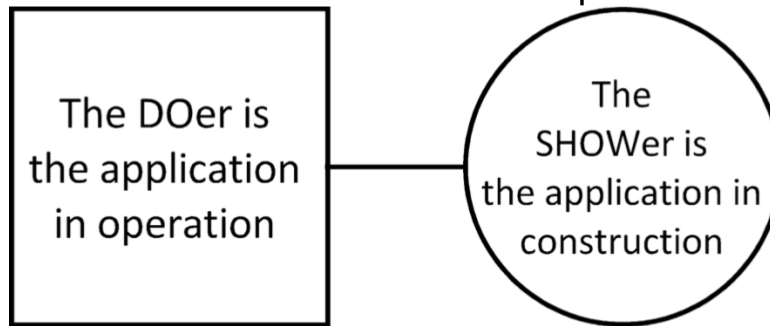
In its simplest form, the life cycle of an interactive business application has two phases.
-The Operation Phase. The object code of the application is executing.
10/39

-The Construction Phase. This is the design/development phase of building. In this conventional example, the artifact (the thing being built) comprises developer-readable source code, from which a software toolset creates computer-executable object code.
11/39

Mapping this onto our pattern we get:
-The DOer is the Operation Phase of the application’s life cycle. The code is executing in an operational environment.
12/39

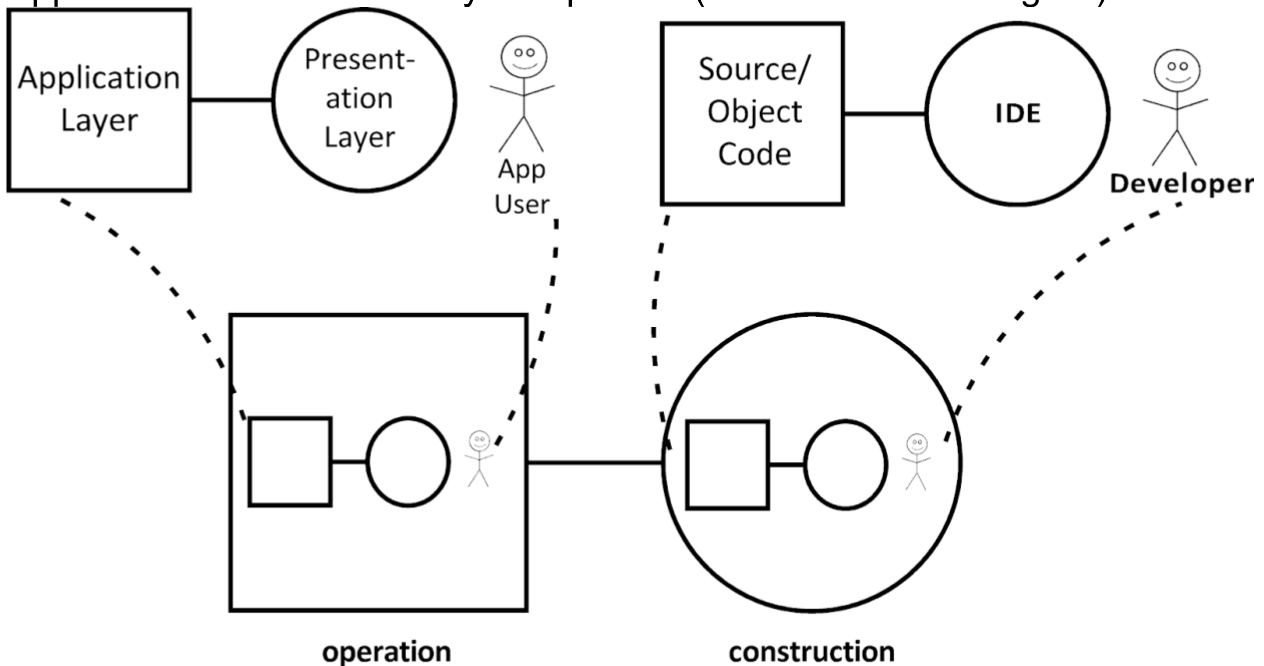
©2020 Melvin Conway

-The SHOWer is the Construction Phase of the application's life cycle. The code is being examined and modified in a development environment.



13/39

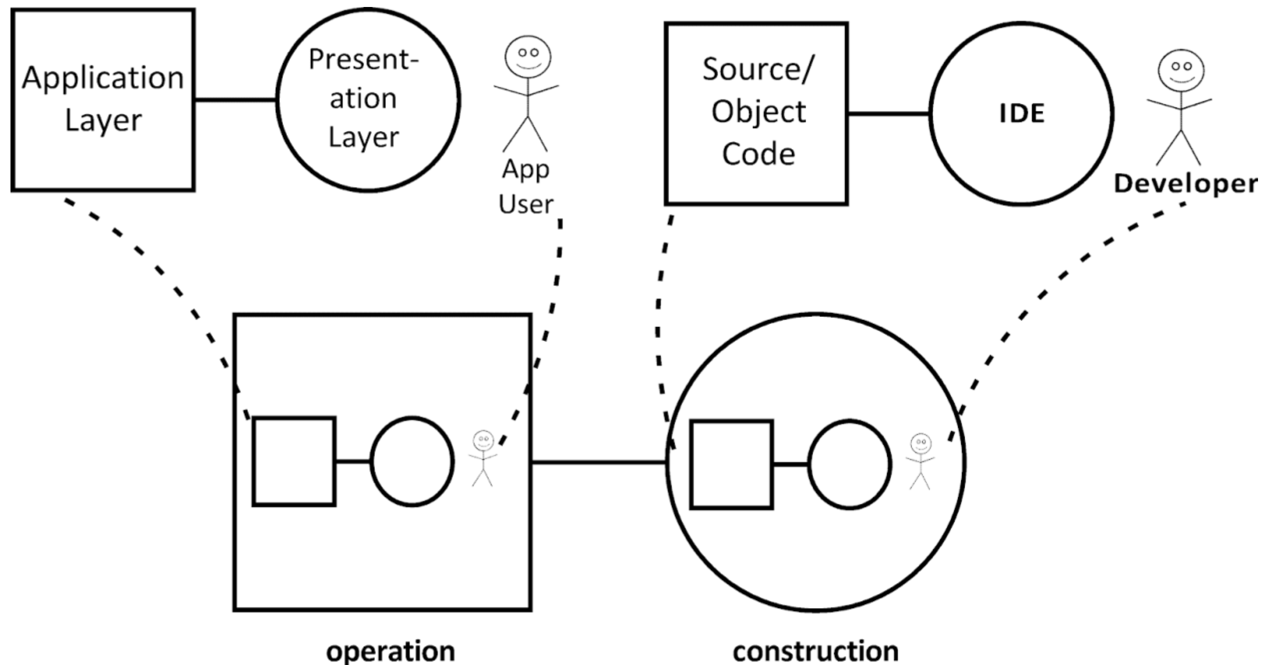
Looking inside the operation phase of the application's life cycle, we can account for the application's GUI with one or more instances of the Application-Presentation Layered pattern (see the left of the figure).



14/39

©2020 Melvin Conway

In the construction phase (see the right of the figure) we will see the developer viewing and editing the source code using an Integrated Development Environment as the SHOWer.



15/39

Pattern Example 4: Self-revealing Parameters

Definition: A Self-Revealing Parameter is a parameter equipped with sufficient viewer(s)/editor(s) for examining and/or specifying it.
(See https://twitter.com/conways_law/status/1238543297561464837)
16/39

The normal way the parameter is specified or examined is through these viewer(s)/editor(s), which conform to Humane Dozen #9, Self-revealing.
(See http://melconway.com/Home/pdf/humanedozen.pdf)
17/39

There is a detailed implementation description of one case on pages 29-30 of http://melconway.com/Working/WP_20.pdf,
Here you see the power of Smalltalk in implementing the pattern.
18/39

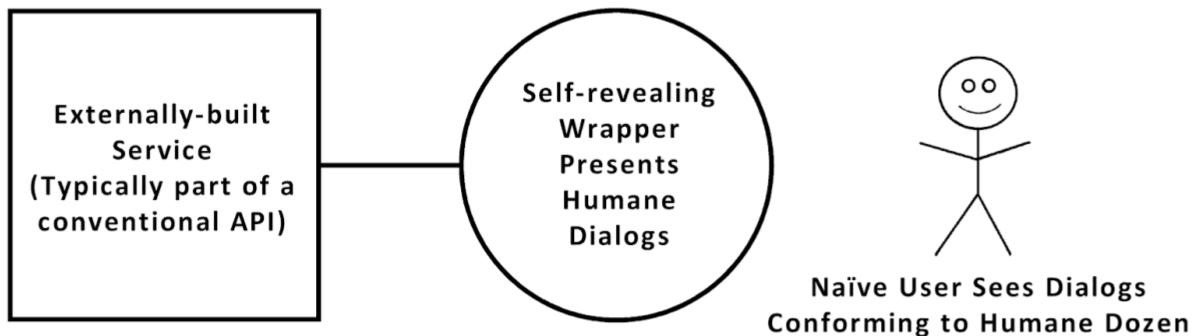
Pattern Example 5: Self-revealing Services

Definitions:
-An Accessible API is a collection of one or more Self-revealing Services.
19/39

-A Self-revealing Service looks like the service analog of a self-revealing parameter. It is a service equipped with sufficient viewer(s)/editor(s) for examining and specifying a request to it.
20/39

But Self-revealing Services are very different in this respect: systems are built by combining externally-built services. Therefore it should be possible to take a conventional service built elsewhere and have a way to turn it into a Self-revealing Service.
21/39

We do this by treating the externally-built service as the DOer of our pattern, and adding a SHOWer to it that appears to the external user as a Self-revealing Service. The SHOWer acts as a wrapper that presents a standardized interface that conforms to the Humane Dozen.



22/39

©2020 Melvin Conway

There is a detailed description from the restaurant use case on pages 31-39 of http://melconway.com/Working/WP_20.pdf,
23/39

Pattern Example 6: Immediate-Turnaround WYSIWYG Development Tools (wTools)

This example applies the Humane Dozen to tool building.
Definitions:
24/39

In
Immediate-Turnaround Development
every change to the source program is immediately reflected in the behavior of the object program. At the very least we need a very fast compiler or an alternative to the edit-compile-link-run-debug cycle.
25/39

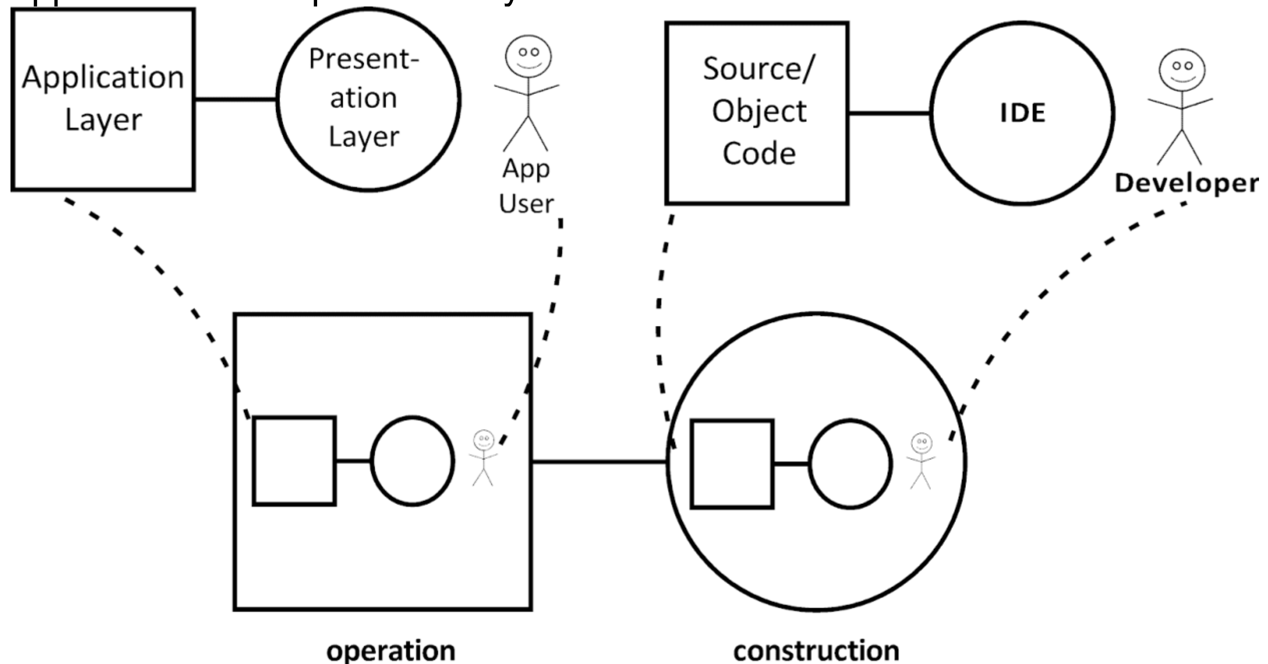
In
WYSIWYG development
we further reduce the cognitive load on the developer with this requirement:
Eliminate the distinction in the artisan's mind between an executable object language and a readable source language.
26/39

Therefore no debugger will be necessary to translate between the two, and no distractions will arise from the need to manage the correspondences between expressions in two languages.
27/39

Here we apply just about everything we have seen above in order to address the question:
28/39

How do you build an Immediate-Turnaround WYSIWYG Development tool? For the sake of brevity, we'll call it a wTool. We need a name for the Application Under Construction, which we'll call the App.
29/39

There are multiple applications of the pattern.
Let's return to Pattern Example 3, which describes the conventional application-development life cycle:

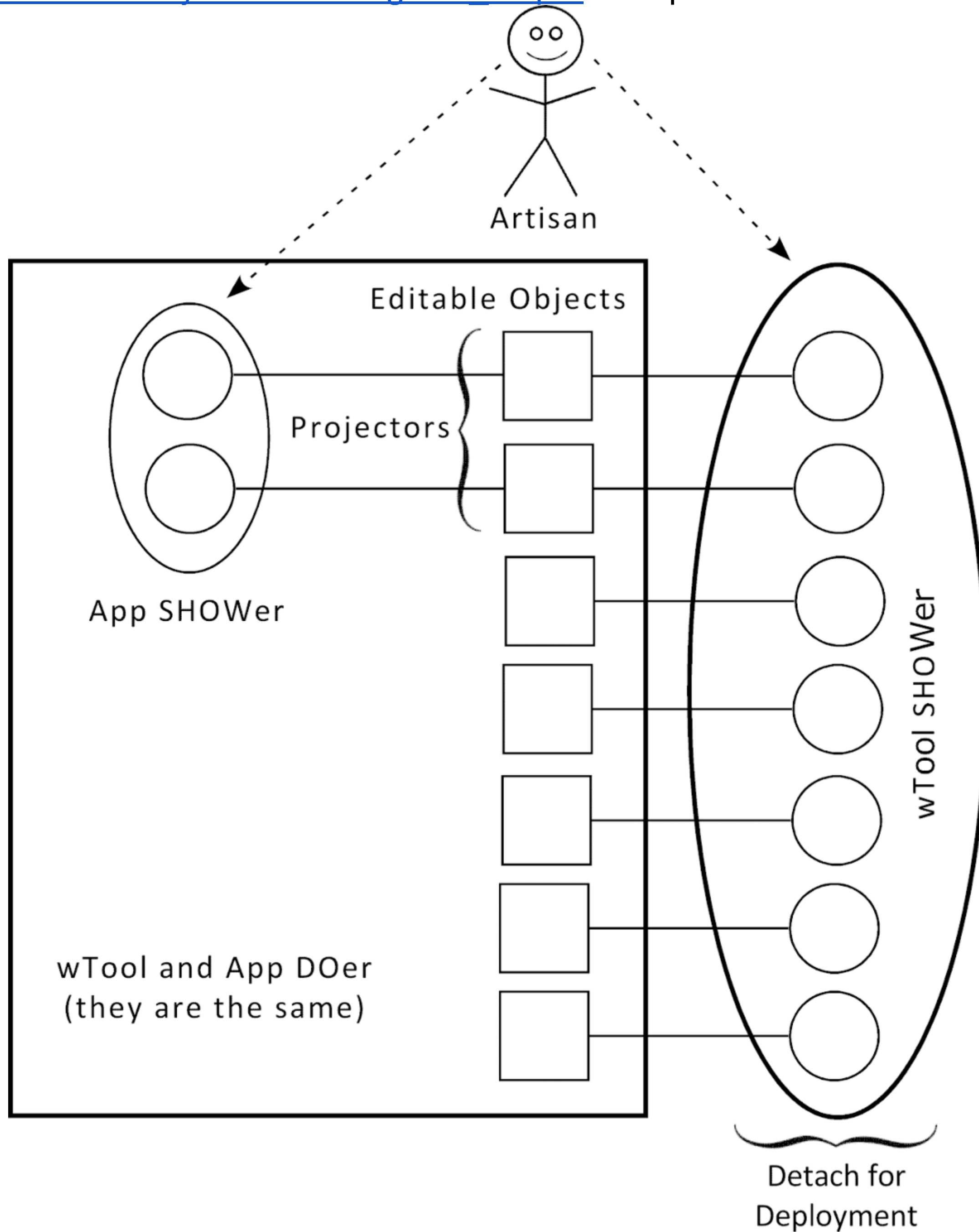


30/39

How does a wTool differ?
1.The Operation and Construction phase of the life cycle (namely, the DOer and SHOWer parts of the wTool) are running concurrently.
2.As you have seen many times above, their SHOWers are projecting their results side-by-side.
31/39

©2020 Melvin Conway

3.The User and Developer are the same person; we'll call her the Artisan. How to represent this? This is an interesting demonstration of the expressive power of the pattern. It also relies heavily on Humane Dozen #8, Isomorphic.
(http://melconway.com/Working/WP_20.pdf is required for some context)



32/39

©2020 Melvin Conway

(A
Projector
Is a component that renders an object on a UI.)

You can see these elements in the walk-throughs in
http://melconway.com/Working/WP_20.pdf:
33/39

•The objects of the executing application (that is, the App DOer) that are reflected in the wTool's UI (that is, the wTool SHOWer) that have behaviors in response to Artisan events in the wTool's UI are Components and Wires.
34/39

•These are the elements of the projection of the App in the wTool's UI for which atomic editing gestures exist.
  -A new component instance can be dragged out from a component palette.
  -A showing wire can be deleted.
35/39

  -A showing component can be deleted after all its connecting wires are removed.
  -A new wire instance can be created (subject to compatibility constraints embodied in the components) by dragging between a source and a sink connector of two different showing components.
36/39

•Completion of any of these gestures triggers a change in the App DOer, which can generate a flurry of internal events. Those four events and their responses define the semantics of the wTool/Artisan interaction.
37/39

•Note that, at the risk of oversimplification, deployment means detaching the Tool SHOWer, and maintenance means re-attaching it.
•The semantics of Artisan events in the App SHOWer are defined by the respective projector components.
38/39

©2020 Melvin Conway

Personal comment: The power of this pattern, in my view, is as a thought pattern. It has influenced the way I think about Humane tools. I intend to be writing more about this.[1] I believe that there are many examples, in multiple domains, that can be added to these six.
39/39

---

[1] Added 2026-01-25: The years of learning that led to this paper are embodied in this presentation: https://melconway.com/Home/pdf/Two-sidedMarketProposal.pdf

©2020 Melvin Conway