Nonprogrammers Can Build Important Parts of Prototypes

Abstract: Here is a two-part whole-application model that enables nonprogrammers who understand the client business to participate importantly in application prototyping.

The main part of the model is an executable skeleton of the application. It describes one or more use cases, including user presentations and interactions, and is built without code using an interactive wiring tool. The skeleton is separated from the business-domain-specific part, which is created by collaborating developers, by an API layer that (1)helps to isolate domain-specific behaviors from the wiring diagram, and (2)is usable within the code-free wiring tool.

This model has promise for building prototypes and monolithic applications in general, but it should be of particular interest for empowering domain experts in Domain-Driven Design teams to participate fully in prototyping.

Twitter: @conways_law

Date of pdf: 7/10/2018
conway.mel@gmail.com

My work has focused on simplification of the conceptual model of interactive applications. Its current results can be summarized in these two parts:

1. Twelve principles of humane application-building language-tool design.[1]
2. The design and preliminary implementation of an interactive wiring-model application-building tool as a usable embodiment of these principles.[2]

Domain-Driven Design Presents a Prototyping Opportunity

In August 2016 Mathias Verraes[3] invited me to speak at DDD-Europe 2017. He introduced me to Eric Evans[4] and I set out to read Eric's book, *Domain-Driven Design*[5], whose thesis is at the heart of DDD-xxx conferences.

I am going to simplify the DDD thesis here for the sake of my present argument:

> To capture the complexity of a client organization in a software design requires a design team in which developers and expert representatives of the client business ("domain experts") are collaborating peers in every respect. In particular they must learn to speak of the details of the organization's behavior, and design the corresponding domain objects, precisely in a language of their own devising and specific to the organization.

Preparation for DDD-Europe led me to think about prototyping as a concrete opportunity for simplification. My assumption is that prototyping is one stage in the progression from design to construction, and quick-turnaround prototypes that have correct function can add value by improving communication between the client organization and the design team, both with respect to speed and accuracy.

---

[1] http://melconway.com/Home/pdf/humanedozen.pdf

[2] See the 6 ½-minute video at https://vimeo.com/275108662 for the most efficient presentation of the wiring model to date.

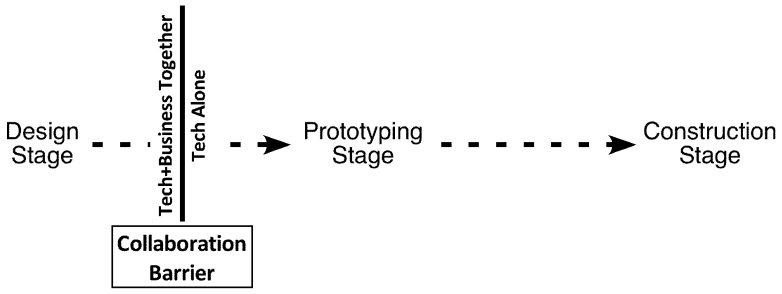
[3] https://twitter.com/mathiasverraes

[4] https://twitter.com/ericevans0

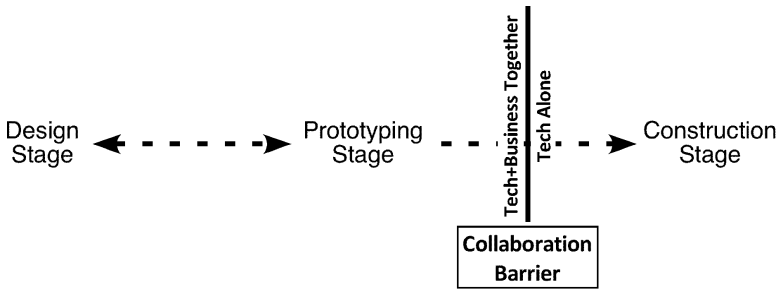
[5] https://domainlanguage.com/ddd/nontechnical-path-through-the-book/

Opportunity: Can the mutuality that exists in the domain-object design stage be extended to prototyping?

Problem: The peer relationship between the carriers of technical knowledge and of business knowledge that exists in the domain-object design stage typically breaks down in the transition from design to prototyping because the business experts don't have the skills to participate in prototype construction as peers with the developers. The collaboration barrier in the figure below represents this obstacle.



Inquiry: Is there a way the wiring model, up to this time only an object of research, can be extended to building usable application prototypes employing real (or approximate) domain objects? Doing so could extend the peer relationships between domain experts and developers into the prototyping stage. This extension is represented below. Note the double arrow that suggests that design and prototyping are a fluid continuum.



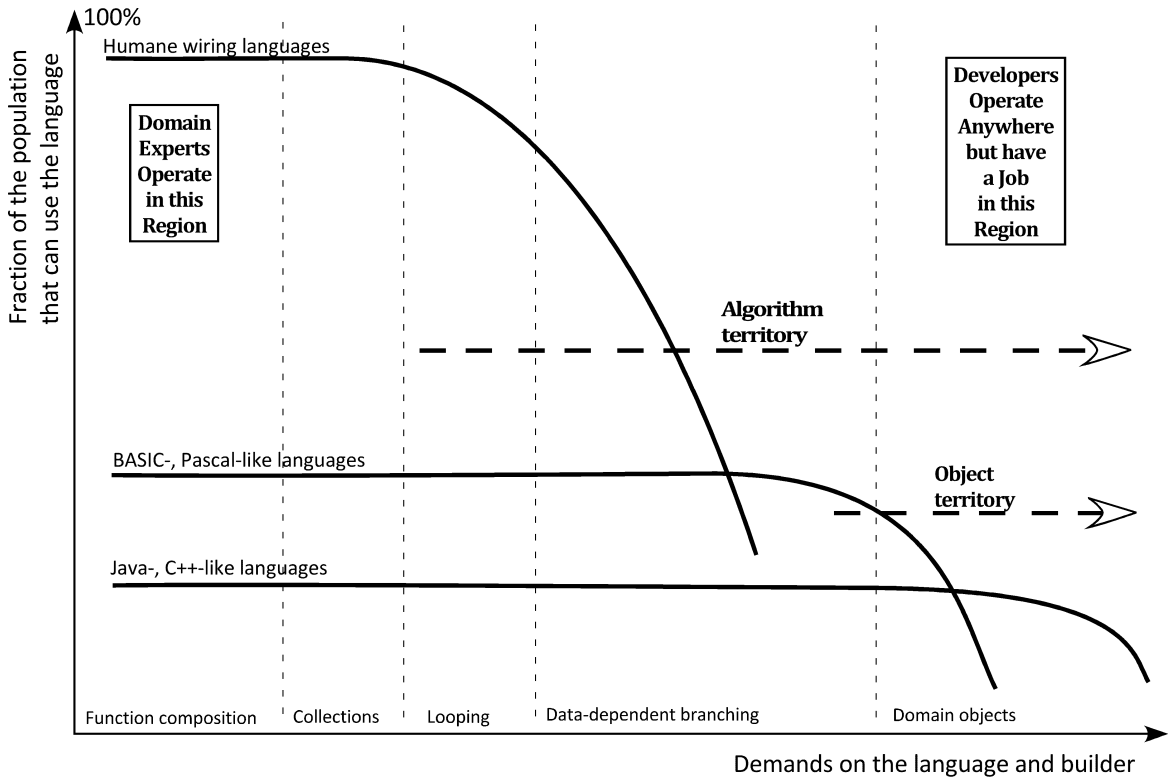
Twitter: @conways_law
Date of pdf: 7/10/2018
conway.mel@gmail.com

The Case for a Two-Part Model

When I built the first version of the shopping cart in my restaurant demo, I needed a way to add up the total prices of all the items. So I built a wired component that iterates over a collection and a wired component that adds. That was the point at which I realized that adding functionality like this degrades the inherent simplicity of the wiring model. I had added iteration and arithmetic and I was on the slippery slope toward an algorithmic language that fewer people could use. I also saw how domain logic was leaking into the wiring diagram.

The following graph qualitatively plots programming language power against accessibility to the general population.[6] It illustrates the futility of trying to meet both of the following two goals with a single language:

1. Everybody can use it.
2. It can do everything.



[6] Apologies to the FP folks; I'm not there yet.

The vertical axis represents the fraction of the population that can use a language. The horizontal axis represents the demands on the language, and correspondingly, the cognitive demands on the users of the language.

The messages from this graph are:

1. Domain experts can do wiring, but once they get into algorithm territory they bog down.
2. Object languages can do everything but domain experts can't use them.

One approach to involving domain experts in prototyping is to partition the building of the application so that:

1. the domain experts and the developers do what they are respectively best at,
2. there is a technical interface between their parts that allows them to cooperate, and in particular,
3. that is consistent with their work in designing the domain objects.

The Two-Owner Application Model

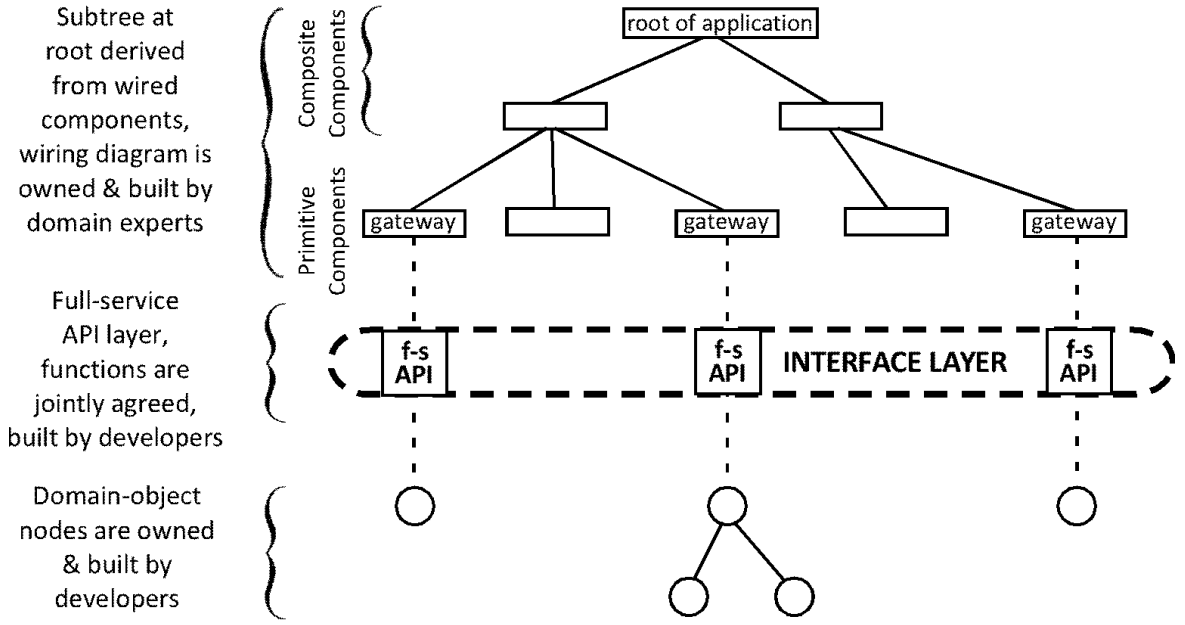
The two-owner application model is my current realization of this approach. It enables domain experts to build prototypes in a no-code wiring language, in collaboration with developers, by connecting certain wired components to the domain objects built by these developers. These "gateway" components are connectors between the wiring-diagram use-case model and the domain-object model. *Their use in wiring diagrams is the means by which the domain experts build prototypes that validate the domain-object design*.

They implement the "full-service API" pattern[7]. Full-service APIs

1. cleanly separate the workspaces of developers and domain experts, and
2. act as a barrier blocking leakage of domain-object behavior into the wired use-case model.

[7] http://melconway.com/Home/pdf/fullserviceapi.pdf

Here is a schematic diagram of the two-owner model of an application.



Explanation of the schematic diagram. This is a structural model. It does not represent certain functional relationships; for example, wiring is not represented. However, if the model represents an actual application in development, each part references some concrete object in some development tool. For example, the circles reference domain object classes, the squares reference full-service APIs, and the long rectangles reference wired components in a wiring tool. *This is an umbrella model that can encompass the tools used both by domain experts and developers.*

The horizontal dotted interface layer separates the worlds accessed by domain experts (above) and developers (below). The full-service APIs connect gateway components and domain objects. These APIs have user interfaces in gateway components within the wiring tool, and developers build them. *They implement the precise domain-object services negotiated within the design team.*

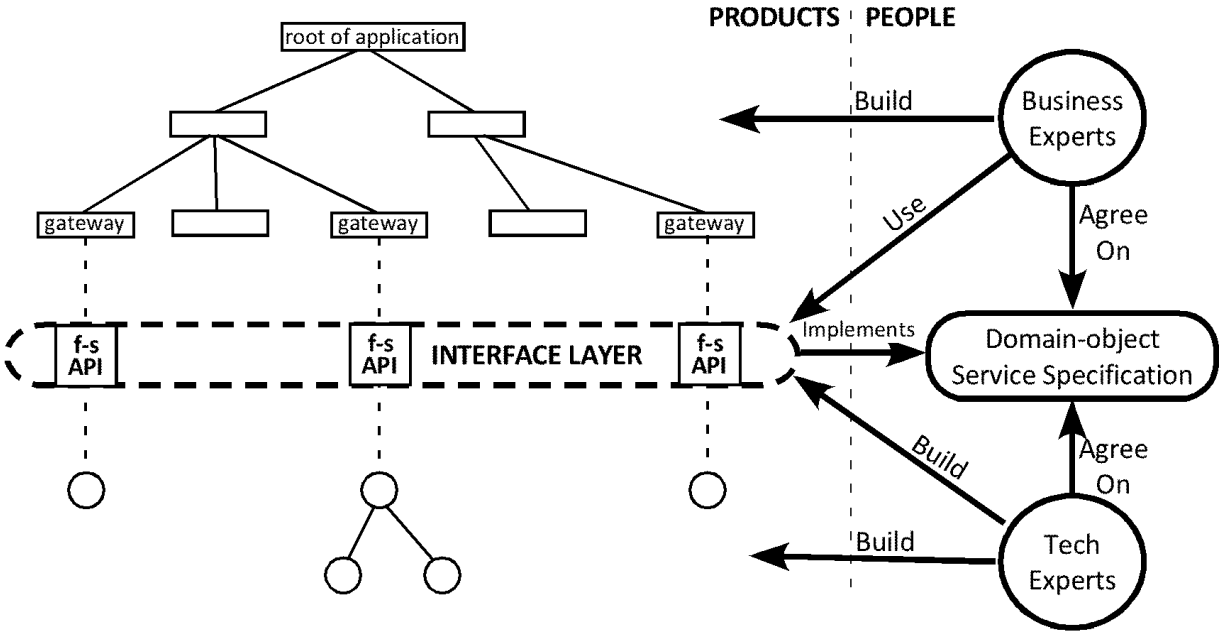
The tree structure above the dotted line expresses the containment relationships among component instances. Primitive components (those built with code) are on the bottom row. Composite components encapsulate the components below them in the tree. The root of the

Twitter: @conways_law

Date of pdf: 7/10/2018
conway.mel@gmail.com

tree is the application itself as seen by the operating environment.

This model can be seen within a larger People-Products Diagram[8] that shows the social as well as the technical structures. The arrows show what each part does.



In the context of a Domain-Driven Design project, the oval labeled "Domain-object Service Specification" can be seen as a direct result of the team's development of its Ubiquitous Language. It is to be expected (and this is the message of the double arrow in the second figure on page 3) that this specification will evolve as team learning occurs. The record of versioning of the specification and its corresponding interface layer will record the learning of the team.

[8] E.g., http://melconway.com/Home/craft2018/008.html

Demonstration

Past: I have been working with a simple restaurant-menu application in my recent conference talks. At the end of the May 2018 Craft Conference[9, 10] talk I demonstrated the insertion of a gateway component as a replacement for a specifically designed component for performing an SQL SELECT.

This demonstration is in two slides.
http://melconway.com/Home/craft2018/034.html
describes the behavior of the gateway component, here called a "Generic Message" component. The next slide,
http://melconway.com/Home/craft2018/035.html ,
shows the removal of the special-purpose SQL component in the demo application and its replacement by the gateway component without a change in the function of the application.

Future: This is, of course, provisional at the time of this writing. My intention for GOTO Berlin[11] is to rebuild from scratch the same application using only basic infrastructure wiring components plus multiple instances of one gateway component. This will show how, in each position of the component in the wiring diagram, the options presented by the component depend on its input.

If I have the time to develop the necessary additional projector components I intend not actually to build the application from scratch but to morph the Three Stooges[12] demo into it seamlessly. If I'm successful at this, it should be a convincing demonstration of the fluidity suggested by the double arrow in the second figure on page 3 above.

---

[9] https://craft-conf.com/speaker/MelConway

[10] Watching the talk is easier with this annotated slide show http://melconway.com/Home/craft2018/ than the video provided by the conference organizers.

[11] https://gotober.com/2018/sessions/568

[12] http://melconway.com/Home/craft2018/006.html