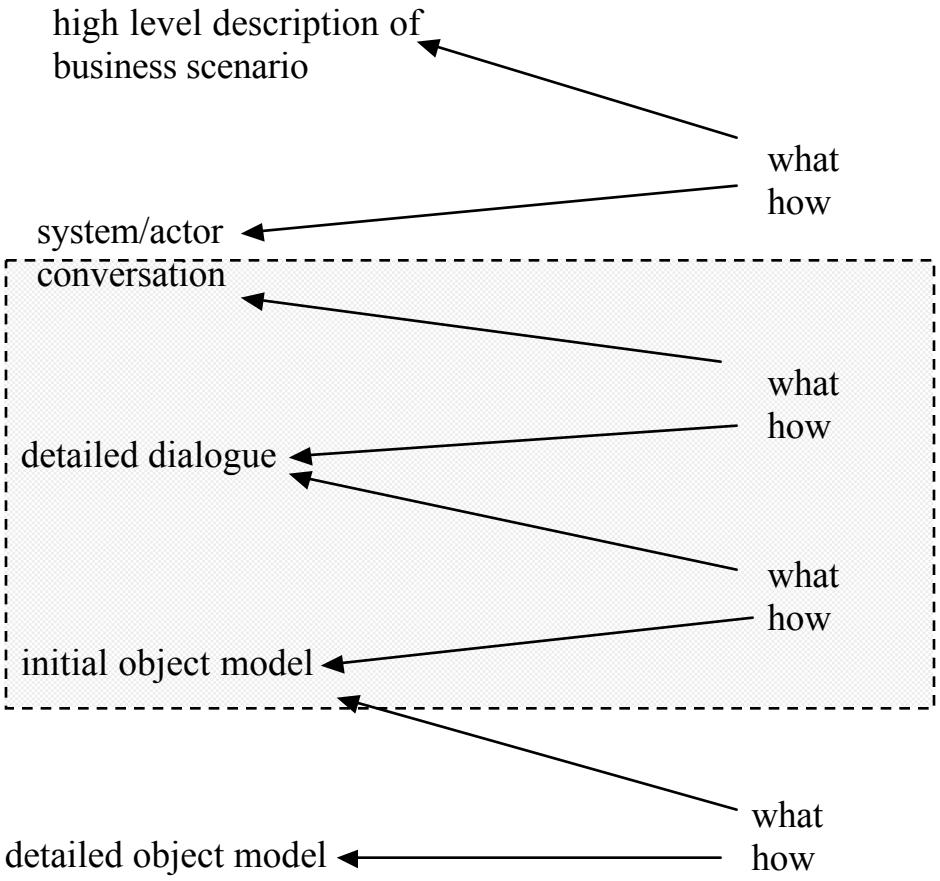
The Art of Designing Meaningful Conversations

Reprinted from the Feb 1994 issue of *The Smalltalk Report*
Vol. 3, No. 5
By: Rebecca J. Wirfs-Brock

In my last column I introduced a framework for developing and describing use cases (see Figure 1). Use cases, scenarios or scripts are roughly synonymous terms for important ways to focus our design activities. In this column we'll see how use cases, system/actor conversations, and supporting object designs fit together. We'll explore some issues surrounding the development of these models. We're still learning about where certain techniques work well and where they fall short. There are several helpful hints and some pitfalls. First, let's briefly review this framework.



A Use Case Design Framework
Figure 1.

Use Case Framework
A high level scenario is a textual description of some process or business transaction that our system must support. Large system development efforts may generate a wealth of detailed process descriptions. Less formal designs still benefit from writing high level descriptions of desirable actor activities.

We prefer to transform these high level descriptions into conversations *before* we model with objects. This forces us to separate out what the actor does and expects from our system's behavior. We purposely leave out details from conversations that should be recorded elsewhere. We omit complex conditional logic from conversations (e.g. if this happens and such and so is true, and x is not, then...) Supporting information belongs in detailed process descriptions or other design documents. We focus on designing the flow of the conversation between the actor and our system.

There are two central parts to a *conversation:* a description of the actor's inputs to our system, and a corresponding description of our system's responses. Together, these 'side-by-side' narratives capture a dialog between an actor and our system. We also list alternatives to the main course. These alternatives represent a reasonably complete list of conditions that object designers must be able to detect and to design appropriate responses for. We also list constraints, timing considerations, reasonable behaviors, and other outstanding issues and desired outcomes (as we think of it) when recording each conversation. Figure 2 shows a sample conversation for Beginning a Session at a Video Kiosk machine.

Conversation: Begin Session

Actors: Regular Customer of Preferred Customer

Overview: A customer identifies herself to our Video Kiosk application. After she has signed on, she can rent, reserve, cancel reservations, or preview movies.

Context: The video kiosk application screen is currently displayed.

Actor Action	**Model Response**
Initiate Session	
	Present greeting Prompt customer for ID
Enter unique ID	
	Receive customer info Validate it

Prompt for permissible actions

Select action

Transfer to selected action

Alternatives:
1. User mis-enters identification
Sign customer on as a guest, after informing her.

2. User's account is overdue by 30 days or more than $50
Inform user and ask if she wishes to make a payment.

Timing Considerations:
Time out and terminate session if user doesn't respond in *n* minutes.

Reasonable defaults/other considerations:
Make it very easy for guest to sign on. They shouldn't have to identify themselves in the same way as other customers. (Note, the conversation for guest's beginning a session may be shown in another conversation, or it may not, depending on how different or important it is.)

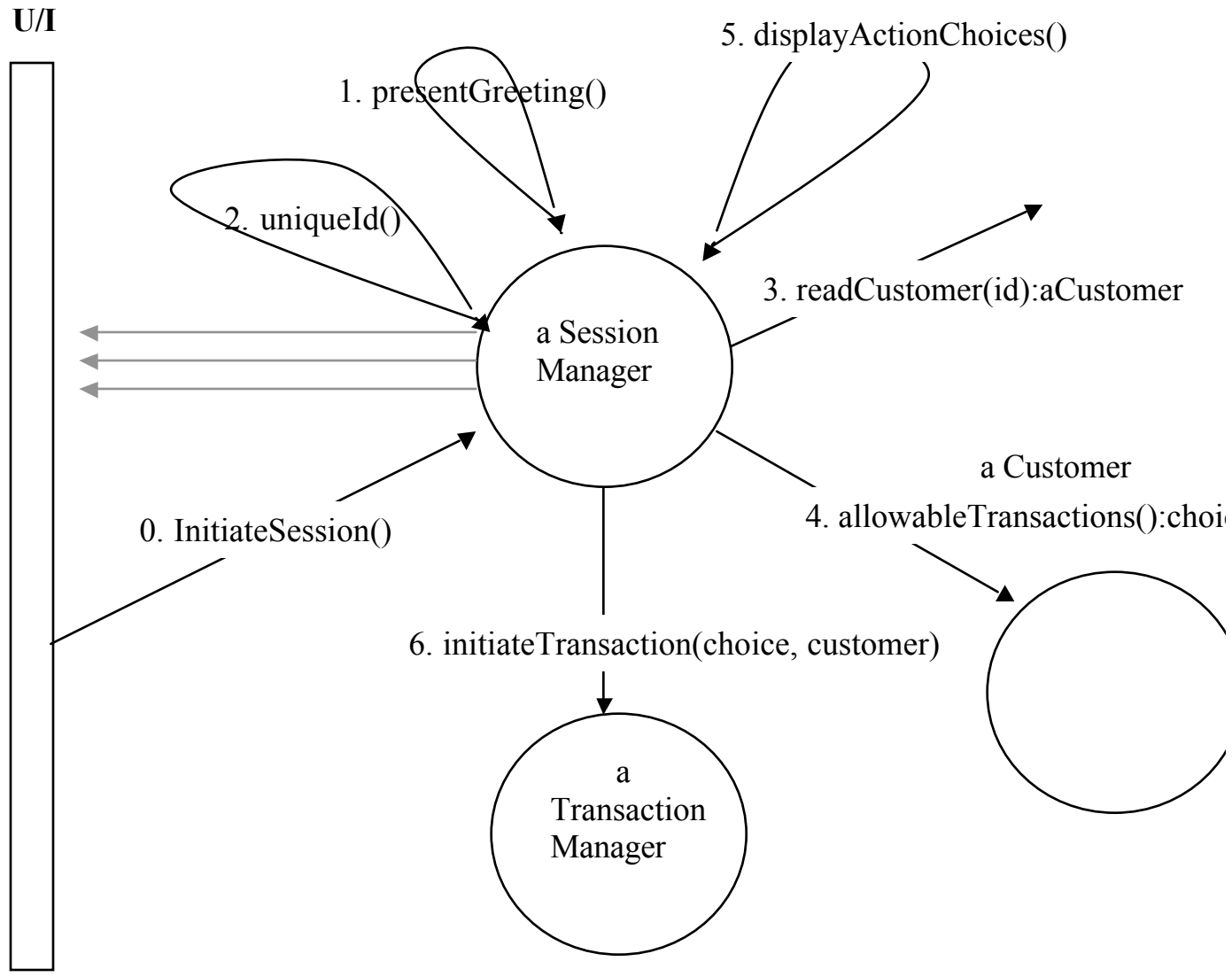
Figure 2. Beginning a Session

Conversations guide our initial object modeling activities. We develop a high level view of key objects and their interactions for each system response side of a conversation. We start by selecting candidate objects according to their roles and stereotypes. If we already have a rough idea of our objects from previous modeling sessions it makes it easier. We still review these partially completed objects and our assumption about them before plunging into modeling. If this is the first time, we obviously spend some time picking out and discussing candidates and their possible roles.

We use white boards a lot during modeling sessions. We record responsibilities and collaborations on design cards. We also draw and redraw potential collaboration sequences, wave our hands, and run through consequences listing pluses and minuses and looking for better ways to distribute object responsibilities. We construct an object interaction diagram depicting what each object does and roughly how it collaborates with others. The results of all this activity are a better understanding of our design and an updated issues list!

Figure 3 illustrates object collaborations for beginning a Video Kiosk session. This diagram shows a collaboration sequence that supports either a regular or preferred customer. Beginning a session involves signing on and picking a transaction from among Renting a Movie, Previewing, and Canceling Reservations and Reserving Movies. Responsibilities were parceled out between a Session Manager, a Transaction Manager, a Customer and a CustomerDatabase object in this modeling session. Since this diagram was entered into a computer, it looks more polished than it

actually should. The collaborations aren't finished. What may look like message names are rough, and more akin to responsibilities than precise message signatures. We've shown sequences of important collaborations, and also shown responsibilities that the Session Manager performs itself (collaborations 1, 2, and 5 in the diagram).



Finding the Right Sized Use Case

Finding the right size for a use case is largely a matter of personal choice. I have carefully laid out a long string of system/actor events and asked colleagues and design students to split them into meaningful units. We all try to follow these guidelines:

look for meaningful sequences that form some nameable sub task

don't look for 'reusable chunks' to start

find an all encompassing name to call this activity

clearly name the sub task using a single action name

look for loops or repetitive event sequences

How people chunk tasks together varies widely. Some people have a hard time justifying their decisions ("it just feels right" *isn't* a justification). It really is a matter of how much or how little detail you can handle, and what hidden assumptions you are making about the underlying complexity of the task. Working in a team, over time, the group as a whole usually develops a collective sense of the right size for a use case. The above guidelines are useful hints, not rules.

Dull Conversations

Sometimes conversations that you generate from use cases are too simplistic. This is often the case with event driven systems. Picking an item off a menu doesn't often generate an interesting conversation either. If you have this problem, look at chunking more tasks together. Perhaps your application isn't very conversational by nature. An actor may initiate a lot of complex tasks for your system to work on, but the interactions to get them started may be pretty straightforward. A command driven monologue isn't that interesting to talk about. Don't force conversations.

Dealing with Slight Variations on a Theme

Actors can be users or other systems. Further distinctions can be made between actors. Some applications I've looked at have dozens of different types of users and dozens of external systems that they have to connect to. Different actors may need or are privileged to conduct slightly different dialogs. Different actors, performing even the same task often need to have slightly different conversations. Wading through all this takes a lot of time. The bottom line, is that unless a conversation or alternative radically differs from a previously documented one, I find it perfectly acceptable to just record the differences. Unless it is that different, I don't bother building a collaboration for each slight variation on the them either. My goal is to produce and build designs, not generate lots of paper to wade through. Consequently, I make sure to record new responsibilities and collaborations on design cards, but don't necessarily show a lot of other supporting diagrams if it isn't warranted.

What Really Goes on During Modeling

I can't ignore big picture while designing the objects for a particular conversation. I don't know anyone who can. Too many side excursions into the weeds can be frustrating. However, remembering consequences of prior modeling decisions and checking for possible inconsistencies is extremely important. Consolidating and unifying design is a constant background mental process that should be allowed to bubble to the surface at times.

For example, if I know what Session Manager should do, I don't park my understanding of its other duties elsewhere while designing how to begin a kiosk session. If I did that, I might add to its pre-existing responsibilities in an inconsistent way.

Showing Model Responses

Typical object collaboration diagrams look either deceptively simple or overly complex. They don't reflect the thought or effort that went into producing them. In order to understand their significance, it is necessary to know what are key objects and what level of detail the designer intended to model. To find this information, either we need to look at supporting documents such as emerging class designs, prior conversations, and other supporting evidence. Designers can also help by walking others through their design. Diagrams alone can't show the significance of key design decisions. This is hard to find in class specifications, too. That's why designers need to tell us what they thought important.

For example, our CustomerDatabase object isn't a simple minded database interfacer. It doesn't just humbly reconstitute customer objects from stored information. Sure, it encapsulates details of some relational database interface objects. But it is really smart. It detects whether a customer is typical, preferred or a guest and dynamically builds the right kind of customer object based on a number of decisions it must make (e.g. is the customer late paying her bill, how many rentals has she made recently, etc.) You can't see those interior details unless we showed the Customer Database object in much further detail on this diagram. However, when we were focusing on the high level objects and their interactions, we didn't want to clutter up our diagrams with too may lower level details.

How Detailed Should an Object Diagram Get?

I used to have a problem showing message sends to self. It seemed like too much detail too early. Now it doesn't bother me if it is done sparingly. Designing an object involves deciding what it does itself as well as all of its collaborations. If you haven't internalized that you need to send messages to self to create well-designed objects, you need to show this detail. If you know this, but your peers still find it confusing, you probably need to show them that detail. Diagram details are a matter of personal discretion, design team standards, and need to comprehend. Either too much clutter or too few details can cause problems.

One important thing to note about our begin session collaboration diagram is that we showed interactions with the user interface as dashed lines. We knew which objects collaborated with the user interface, but we didn't know how they collaborated. *We purposefully didn't show this*

detail. We did this for several reasons. We wanted to first concentrate on the interactions between business objects, controllers, service providers, information holders, and the like.

If we ignore the interface details for just a bit, we can plug our design into a variety of different user interface solutions after understanding our object model. We also wanted to give ourselves the freedom of exploring potential user interface designs as a separate activity.

We also didn't want to overly constrain our design with user interface requirements. Employing this tactic prevents us from embedded detailed quirky understanding of particular user interface objects. I personally don't have a big problem with setting aside the user interface details until I know what the other objects generally have to do. I advise you to worry about the details of that *after* the interplay between other objects settle down.

Following this strategy causes some feedback and tuning of both our conversation and the objects that interact with the user interface when we actually do attend to those details. If we limit the number of objects that actually interact with user interface objects (which is always a good design principle to follow), the impact of this fine tuning can be minimized.

In Conclusion

There are open issues about use cases, conversations and object modeling. However, there is a big payoff in designing this way. Use cases and conversations to guide our design activities. Tying our model back to conversations, use cases and supporting documentation is one big step in the right direction. My goal in applying these techniques is always to enhance my abilities to communicate with non object experts and improve my abilities to produce the right design solution to the right problem.