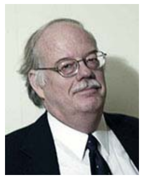
Practical Issues in Cognitive Modeling for UI Design



David E. Kieras

University of Michigan

Powered by Naval Research TM/SM

Introduction to the Session

You've chosen and learned an architecture and its tool.

It's time to start building and using some models.

What are the practical obstacles?

There are many that apply to all architectures and tools.

This session will discuss some critical issues.

Possibly controversial, but should still be useful.

Programming the Simulated Device

The Real Bottleneck in Model-Building

Useful Distinctions

Useful Distinctions - continued

Special-Purpose Device Specification Languages

The Real Bottleneck in Model-Building

To build a model that involves a fully interactive simulated device, the device must be programmed to mimic the interface being designed.

Can be redundant with actual system development effort.
Can be a nasty programming job.

Appears to be a more serious modeling bottleneck than programming the simulated user!

Plan on significant effort before user modeling can begin.

May need excellent programmer on the team.

Future possibility: Being able to tap into an intact application.

St. Amant's work.
However, won't help if system hasn't been built yet - which is when the modeling is supposed to be most useful.

Useful Distinctions

Distinguish between:

What the device has to provide to the simulated human.

What the device has to show the modeler.

In many systems, the input/outputs to the simulated human are simply abstract events.

Visual object appears at (x, y) with color: red.

Mouse cursor moved to (x, y).

No actual GUI or display needs to be programmed

Showing the modeler a visualization of the interaction is a great development and demonstration aid!

Extreme case: Midas

Much harder to build than minimal simulated device.

But could be crude and still support development and demonstration.

- Approach used in EPIC software.

Useful Distinctions - continued

Distinguish between:

Detail needed to support the design decisions.

Irrelevant detail.

Device only needs to simulate:

What the simulated human will act on.

Only at an adequate level of detail.

E.g. don’t simulate in a display anything that the simulated human won't use.

Add detail only when required by modeling needs.

Sounds simple, but takes clear goals and discipline.

- Approach followed in GLEAN modeling of CIC tasks.

Good design of device software enables incremental upgrades in detail.

Special-Purpose Device Specification Languages

Constructing a simulated device can be easy using a specialized programming language.

When is a special-purpose device specification language a good idea?

If the modeling system already provides it, and it is adequate for your problem.

In a specialized domain where many design variations of the same type of device need to be explored.

- Best case: Simulated device is the actual interface system.

  Building the interface also builds the simulated device at the same time.

But experience seems to be that special device specification languages eventually acquire general-purpose programming language facilities.

E.g. MicroSaint’s Pascal-like language has trig functions.

Result of building models for a variety of domains.

Fate of special languages:

Almost as complex as a standard programming language.

Modeling system developers must develop, document, maintain, and support with editors, debuggers, bug fixes - time away from psychological model support.

Must be learned by the modeler, taking time away from psychology as well.

Better choice for modeling system developers:

Allow modeler to use a “dummy” default device if it will work.

Allow modeler to program a device easily in a standard programming language.

- Learn and re-use skills from education and standard sources.
- Good support often present in the modeling system implementation.

Identifying the Task Strategy is Critical

Why the Task Strategy is Critical

Identifying the Strategy used in a Task

Heuristic for Design: Model what Users *Should* Do

Why the Task Strategy is Critical

Human performance in a task is determined by:

- Requirements of the task.
  What the human is supposed to accomplish.
- The cognitive architecture.
  Perceptual/cognitive/motor mechanisms.
- Strategy for doing the task.
  Given requirements and architecture.

Performance in a very wide range of tasks is heavily influenced by strategy.

Low-level simple dual tasks - psychological refractory period.
Middle complexity procedural tasks - telephone operator.
Higher complexity dual tasks - aircraft pilot.

Architecture contributes nothing without a task strategy.

Like a computer without programs - doesn't do anything.
The strategy specifies how the architecture is used.
Model validity depends on correct strategy as much as correct architecture.
No strategy, no predictions.

Model construction and application require:

- A specified cognitive architecture.
- A representation of task strategy.
- A methodology for devising strategies based on task analysis.

Identifying the Strategy used in a Task

Key input to a model is the strategy that the human follows in the task.

Often, the strategy is not obvious, even to "experts."

Even highly experienced people don't always know or use the best procedure.

- Even trainers often do not know.
- Procedure training materials often suboptimal.

How does one identify the task strategy?

Cognitive psychological research.

Very slow and expensive.

Often not scalable or appropriately generalizable.

- Artificial lab situations, or restricted sample of natural situations.

Knowledge engineering.

Usually works well enough for a useful system.

Very slow, very expensive.

Not necessarily realistic of human performance.

Task analysis using Human Factors methodology.

Well developed, long history of application.

Can be fairly slow and expensive - have to learn about task, and observe users.

Not very rigorous - methods are actually informal.

Not well related to models of performance.

Heuristic for Design: Model what Users *Should* Do

Given complexities, how do we know what users do, or will do?

Answer: You don’t, and it is too hard to find out!

E.g., people can under-use a complex system to an amazing extent!

Best approach: Model how interface should be used.

Design for simplicity, consistency, speed of performance assuming system is used as intended - best-case analysis of value of features and interface.

- If resulting system is too slow or complex, it will definitely be inadequate when used by less-than ideal users!

Separate issue is whether users can or will use it that way.

- Trainability, learnability, quality of training materials and program.

Bracketing logic can help evaluate consequences of user choice.

Compare full versus minimal usage of design features.

Compare fastest-possible to slowest-reasonable strategy.

If bracket is above or below requirements, answer is clear.

If bracket straddles the requirements, we have a problem to resolve.

Favor Generative Models

What is a Generative Model?

Value of Generativity

Some Generative Modeling Systems and Methods

What is a Generative Model?

A model is generative if a single model can generate predicted behavior for a whole class of situations without having to be rewritten or rebuilt.

Each situation or scenario is input data to a model that then generates the appropriate predicted behavior for the situation.

Implication: Scenario does not list the human's behavior, but lists data about the external objects and events in the situation.

The stimuli, not the responses, though can be contingent on responses.

Contrast with a model that has to be rebuilt for each different scenario.

Beware of any analysis method that includes specific action sequences for a specific situation.

- E.g. typical Keystroke-Level GOMS model.
- Often fails to transfer to different, or more complex, situations.
- Often too labor-intensive to apply to multiple scenarios.

Value of Generativity

Once a generative model is constructed, usually can run any number of scenarios easily - just computer time and results analysis.

Important for complex models or scenarios, critical systems.
Design coverage requires exploring multiple scenarios, not just one.

Generativity provides easy design iterations.
If system design is modified, usually model requires only a small modification, and the predicted behavior can be easily regenerated.
But initial construction is typically harder than a non-generative model.

Generative models typically directly represent the user's procedural knowledge - what the user has to do.
E.g. NGOMSL shows "how to do it".
Can often be inspected to see how the design “works”.
For example, are similar parts of the task done with similar procedures?

Some Generative Modeling Systems and Methods

Some human factors methods.
Hierarchical Task Analysis (HTA) - procedures and plans.

Task network models.
Scenarios are simply different input event sequences.

Most cognitive architecture models.
Programmed with procedural knowledge on how to respond, not a specific response script.

Some forms of GOMS models.
Those that have explicit procedures, rather than specific action sequences.

The Role of Design Detail in Modeling

Can Design Detail be Avoided?

Modeling Systems that Require Detail

Task Network Models Require the Least Detail

Can Design Detail be Avoided?

Modeling requires developing and representing detail about the design.

Especially if the perceptual-motor aspects are going to be captured.

- Where is each button, how big is it, etc.

Often cited as a problem with modeling - developers don't want to get bogged down in all the details.

But interface design is a matter of getting the details right.

Research results: The quality of the design at the detailed level is more important than even the interface style.

So at some point the details have to be designed and evaluated anyway - just a matter of doing it at the best time.

Who creates the detailed design?

Don't leave it up to programmers.

Designer should specify it, and then it is available to the modeler.

Modeling Systems that Require Detail

GOMS and Cognitive Architectures are inherently committed to detail.

These work at the level of detailed psychological models, so require the most detailed input.

- E.g. Fitts' Law requires location and size of each button in the interface.

Psychological constraints usually depend on details of situation.

Means hardest to set up - considerable input required to get any modeling results.

Modeler can get tangled up in detailed issues where the correct psychological interpretation is not yet available.

A serious risk with Cognitive Architectures, less so with GOMS family.

Example: Modeling use of radar display: How is visual memory involved? How does it work? What are the limits?

More later.

Task Network Models Require the Least Detail

No need to commit to any interface design at all!

Each task in the model can be anything at all, with any desired set of characteristics!

If you can choose the characteristics without a specific interface design in mind, you can get some model results.

Typically used in early stages of military system design:

E.g. how many crew needed for a new attack helicopter?

Choose mission profile.

- Like a high-level scenario of what has to be accomplished by the new system.

Do functional analysis.

- What functions need to be done?

Do a function allocation as the first top-level design.

- What done by humans, what done by machine?

Build and run a model to see if the workload requirements are reasonable, and performance satisfactory.

No point in using a detailed model before making fundamental decisions on functions or number of people!

So task networks are the best (or only) choice for early design.

Positive effect of the lack of psychological constraints.

Doubtful that it is the best choice for modeling detailed designs.

Concerns over Model Validity

Can You Believe the Model?

What About Gaps in the Psychology?

Should You Validate the Model?

Should You Validate the Model - continued

Should You Validate the Model - conclusion

Can You Believe the Model?

Suppose your model implies critical design choices. Can you believe it?

Poor response: Build and test the same prototypes that you might have built without the models.

Means the modeling was a waste of time, so don’t bother with it the next time.

Better response: Understand how the model implies the design choices.

What aspects of the model are contributing to the outcome?

- Discover by profiling the model processing, analyzing the model structure.

If relevant aspects are known to be valid, then result should be accepted.

E.g. design is unacceptably slow because user must work through many menus.

If relevant aspects are problematic, then result must be checked out.

E.g. model assumes that user can remember information about all previously inspected screen objects, and so does not need to search for them again.

- Problematic because bounds on visual memory are unclear.

Sensitivity analysis can reveal whether design choices depend on assumptions.

E.g. Modify the model so that number of screen objects remembered is small.
Does this change which design is better?
If not, then problematic issue doesn’t matter.
If so, then you need more data on real performance or a different design.

What About Gaps in the Psychology?

Gaps in psychological constraints:

All of the modeling systems are psychologically incomplete and approximate and will remain so for a long time.

- Must be prepared for problems and puzzles to arise during application.
- An advantage of GOMS: Might be crude approximations, but at least not as much psychological detail to distract the modeler.

If model results depend on an problematic issue:

Use bracketing logic or sensitivity analysis: Can design choices still be made?

Resolve in favor of less human capability.

- Conservative, appropriate if user will be under stress.

Resolve in favor of what seems intuitively correct about the task.

- Can still be better than not trying to model at all - just be sure not to hide it!

If data or theory is available, use it to modify or supplement the architecture.

- Special case "kludge" if architecture can't be changed.

Should You Validate the Model?

Remember that testing with real users must be done sometime.

Models don't cover all of the design issues.
Final user test can reveal other problems, and also modeling mistakes.
Comparing final test data to model might be useful.

But should you collect special data to validate the model before use?

"Let's be sure model is correct before using it in the design!"

Validation is not a "normal" part of model usage:

A task for developers of modeling methodology, not end users of it.
Serious validation data needs to be much more controlled and detailed than normal user testing data - very hard to collect.

If you have the resources to collect and analyze validation data, do you need the model? Can't you just do lots of conventional user testing?

Might need data for calibration or parameters for later modeling - different problem.

Should You Validate the Model - continued

While a model is certainly wrong at some level, collecting empirical data on human performance in complex tasks is not a royal road to certainty!

Complex real-world tasks involve subtle user strategies, team interactions, influences of background knowledge, and the specifics of scenarios.

- Such experiments are extremely slow and expensive, even with small samples.
- Not practical to run experiments using many scenarios, every reasonable design variation, every candidate team organization.
- May not be able to understand why people did what they did - asking them is usually ambiguous at best, and responses might be idiosyncratic.
- Not necessarily easy to see what the experimental results mean, and whether they are valid and generalizable.

Data can invalidate a model only if the data and the model are about the same situation!

E.g., if test users don't follow the strategy in the model, and the model is based on what people *should* do in the task, then the data is "wrong", not the model!

Should You Validate the Model - conclusion

Instead of validation attempt, understand what the model is doing and why:

Is the model strategy based on an analysis of what users*should* do?
If not, why? Does this make sense in a design context?

Do you believe users can or will follow the same strategy as the model?
If not, why? Is it a training issue, or is the model simply wrong-headed?

Is the model plausible and non-problematic in its assumptions?
If not, see earlier suggestions.

If all answers are "yes", then the model results can guide design decisions without special validation efforts.
Of course, this might be a mistake, but modeling is only useful, not perfect.

Summary

Several practical issues were summarized:

Programming the simulated device.

Identifying the user's strategy.

The value of generative types of models.

The role of detail in modeling interfaces.

Concerns over model validity and how to deal with them.