# Accountable Modeling in ACT-UP, a Scalable, Rapid-Prototyping ACT-R Implementation.

**David Reitter (reitter@cmu.edu)** and **Christian Lebiere (cl@cmu.edu)**
Department of Psychology, Carnegie Mellon University,
5000 Forbes Ave, Pittsburgh, PA 15213 USA

## Abstract

ACT-UP is a toolbox implementation of the ACT-R cognitive architecture, aimed at allowing rapid prototyping of complex models. With ACT-UP, we propose *Accountable Modeling*, where the only model components that are specified are those supported by empirical evidence and part of the model's theoretical claims. ACT-UP is a library providing a programmatic interface to the classical ACT-R functionality. Implemented in a functional programming paradigm, models are reusable in other contexts. The toolbox is demonstrated using five implemented and evaluated cognitive models.

**Keywords:** Complex Models, Cognitive Architectures,ACT-R

## Introduction

Cognitive models have explained a great deal of behavioral and neurophysiological data. On the road to understanding the mind, cognitive architectures have specified a core set of representations and mechanisms common to a variety of models in order to separate general functional components and their abilities from domain-specific instantiations, such as knowledge and strategies. However, the tasks that classical cognitive models have taken on are mainly those that can be defined in a controlled environment. Process models of laboratory behavior are often overly specific and needlessly complex, while alternative models would yield similar fits. The model eco-system has diversified rather than converged, with specific rule sets developed for each given task and very seldom reused or generalized for other tasks. This leads to overfitting and lack of robustness. To robustly explain and predict behavior in complex real-life situations, model complexity has to increase further. Inevitably, humans execute much more complex tasks as well, drawing from a variety of knowledge and skills and contextualizing their observations and thoughts in light of both long-term experience and recently acquired knowledge.

The greater complexity of tasks may have a welcome effect on cognitive architectures. Current general architectures such as ACT-R (Anderson, 2007) or SOAR (Laird & Rosenbloom, 1987) are not as restrictive as human memory is. ACT-R, has, during versions 2 through 4, become more and more restrictive: large, very complex rules made way for smaller, granular ones that could describe less functionality each. Still, even its latest incarnation can implement a model that predicts excellent human performance at the most intricate N-back task, failing to explain the dismal human performance scalability at this task. This difficult task (Kirchner, 1958) requires subjects to keep a first-in-first-out queue of $N$ items in memory. Sufficient architectural contraints may mean that modelers can no longer design functional models of existing tasks, let alone the more complex ones we have argued for. One solution to the dilemma is to constrain models by re-use of micro-strategies. It is hoped that the resulting convergence will eventually let us better reflect the architecture of the mind (Newell, 1973).

As task complexity increases, a careful analysis of the components of the model is necessary. Every rule, every data structure, and every knowledge access process can be seen as a claim that needs to be proved empirically. For anything but the simplest cognitive models, many of the procedures and data structures they define are often not evaluated: the specifics of many of the components of the model may be irrelevant to the story a model has to tell. The solution to this problem is *under-specification*. In what we call the *Accountable Modeling* paradigm, we suggest to apply Occam's razor and specify only what is meant to be directly or indirectly evaluated.

As a consequence, we arrive at models that can be more complex yet faster and easier to prototype, while still using the same core representations and mechanisms of the architecture. Until all portions of the model are fully specified, such models may fall short of Newellian *complete process models*. Yet, they honestly separate claim from conjecture and provide the same level of comparison to human data.

## Accountable Modeling

Recent work has been undertaken to investigate the use of ACT-R to study the interaction of two, eight, or even thousands of cognitive agents. Scalability in this domain would make cognitive models applicable to new domains such as network science, for which a precise computational representation of human cognitive processes has been desirable but as to now unavailable. The modeling methodology in this paper follows *accountable modeling* within the ACT-R theory.The "Adaptive Control of Thought–Rational" framework (ACT-R, Anderson, 2007) defines a component-based architecture, in which specialized modules work largely in parallel to contribute to thought processes. In recent computational implementations, it requires *end-to-end* models, describing thought processes through a set of production rules controlling the interaction of cognitive (e.g., long-term memory) and perceptual components. We distinguish the ACT-R theory from its canonical implementation (ACT-R 6, Bothell, 2005). In the following, we assume familiarity with the basics of ACT-R.

Working within the ACT-R theory, we designed a new toolbox instantiation of the theory called *ACT-UP*. ACT-UP reflects ACT-R, but lets the modeler specify algorithms much like a programmer would. Functionality is compartmentalized in reusable functions (taking arguments and returning a value) and data is stored and retrieved as in ACT-R in chunks in declarative memory. ACT-UP is intended as a library for

modelers comfortable with basic programming paradigms.

Most of the cognitive functions that ACT-R makes available correspond to the *buffer*-based interface of the architectural modules in ACT-R: a *learn-chunk* function to commit a chunk to memory or boost its activation; a *retrieve* function to request a chunk from declarative memory, taking hard constraints, cues (to spread activation), and soft constraints (for partial matching). (In ACT-R, buffers represent interfaces between cognitive modules. Chunks are bundles of feature-value pairs, which can be stored temporarily in buffers, or, more long-term, in declarative memory.) But ACT-UP also makes more fine-grained cognitive functions available. Such micro-functions allow models to go beyond what is available to ACT-R models.

We intend to address several goals with ACT-UP. *Accountability* suggests to underspecify model components that are neither motivated by data or theory nor subject to empirical evaluation. *Rapid prototyping* allows modelers to quickly build and modify most parts of the model, even computationally complex ones, while focusing on learning and other cognitive effects predicted by ACT-R's theoretical assumptions. Crucially, it produces models that are reconfigurable so that systematic parameter search can be used to explore the space of possible models. *Reusability* results from clear input and output data structures, turning models into functions that can be re-used in other contexts: the *convergence* of models and cognitive frameworks is a long-term goal. *Scalability* allows models to run longer, apply to more complex tasks, and simulate agents in the context of larger multi-agent systems.

To describe the notion of accountability, let us consider some design decisions that a modeler has to make: where to abstract away from subtasks and surrounding tasks, and where to concentrate on the cognitive properties that ultimately explain variance in the data. Both ACT-R 6 as well as the ACT-UP toolbox allow for free computation outside the theory[1]. Even the more theoretically motivated buffers and chunks are storage means that are not limited in size. ACT-UP retains information in local variables, thereby acknowledging the lack of constraints.

Procedures are at the core of ACT-R. They initiate perceptual acquisition, declarative memory retrievals and motor actions and act as an information broker between all components of the architecture. They are implemented as *production rules*, defining a precondition that refers to the state of buffer contents, and a consequential action affecting the buffers and their associated modules. While all rules are eligible to match at all times in a model, only one of them is selected to fire.

Such a production rule system is capable of implementing complex algorithms, especially with the addition of state information in buffers. Thus, the question of whether a solution to the experimental task can be formulated as a set of production rules is less relevant than the question of whether the model's crucial decision-making can be cast as a pattern-matching task, or whether reinforcement learning of recognition patterns and associated actions (as in ACT-R's learning of production rule utility) can explain the observed data. Indeed, in typical models does the deciding learning effect occur only in very specific decision-making moments. The large majority of the model's production rules are in place to deterministically execute the task. These collections of production rules are difficult to develop, inspect, change, maintain and re-use. Therefore, ACT-UP's rules may be underspecified and implemented as a program. This will also often be the case whenever productions implement deterministic and static processes. Other productions may still be faithfully described: those that reflect the crucial pattern-matching tasks and reactions to recognized patterns, or the routinization of initially declaratively memorized processes. This is where the toolbox approach allows modelers to underspecify the model by reformulating productions in a more direct, computationally treatable manner. Underspecification may also occur for many methodological reasons. Data may be lacking to support an evaluation of the claims, if they were specified, or the lack of suitable data, or the task complexity, e.g., understanding of complex natural language instructions where it does not reflect the goals of the modeling work.

ACT-UP provides high-level interfaces to core simulation components of human cognition (e.g., *retrieval of a declarative chunk from a pattern specification*). It also gives modelers fine-grained control over such processes, by filtering chunks from declarative memory or choosing the most active chunks from a set. Thus, the functional toolbox approach integrates well with cognitive mechanisms that do not yet have a well-specified interface to the remaining buffer- and productions-based ACT-R architecture.

Wherever constraints are relaxed, cognitive plausibility comes into question. Traditionally, models have relied on their within-theory specification to provide constraints promoting cognitive plausibility (usually using ACT-R 6). As argued in the introduction, such constraints are not exhaustive. In order to constrain the computational resources available to the model, ACT-UP asks the modeler to focus on the crucial portion of their model, while using the computational power of a programming language for other parts. For those parts, parameters may be fitted that describe their (human) execution time and reliability. Since production systems are Turing-equivalent, we know that a production system can be defined to accomplish what an ACT-UP model does. Thus, ACT-UP models do not represent an implausible gain in power.

## ACT-UP architecture

ACT-UP aims to implement a substantial subset of the ACT-R theory. The striking differences between ACT-UP and implementations such as ACT-R 6 or Stewart & West's (2007) Python variant do not lie in the theory: they pertain to the interface that is offered to the modeler. ACT-UP's interface is synchronous and does not yet implement parallelism (which is often not needed). ACT-UP is a toolbox providing ACT-R functionality in a piecemeal fashion as well as commands at a higher abstraction level, which would integrate well with Salvucci & Lee's (2003) motor, speech and perceptual mod-

[1] "eval" statements in ACT-R 6, for instance, allow the modeler to design model components in Lisp.

ule commands. The ACT-UP library is a stand-alone system, and independent of ACT-R 6. It provides a set of Lisp functions and macros; modelers interact with it on the basis of source code that follows Common Lisp syntax (see below for examples). ACT-UP models predict the two major behavioral outcome types: choice and timing.

### Declarative memory system

ACT-UP's declarative memory (DM) embodies all the core elements of DM in ACT-R. Memory is accessed in the form of *chunks*, which are sets of feature-value pairs. Chunks are learned (or reinforced) with an explicit command; there is no automatic learning (*buffer clearing* in ACTR 6). Retrieval occurs with a (normally) synchronous command, in which the model specifies hard constraints (a set of feature value pairs), soft constraints (subject to partial matching), and a set of chunks as cues that spread activation. Thus, modelers gain better control over the context of the retrieval. In ACT-R, buffer contents that can spread erroneous activation have to be tightly controlled (or parameterized) in order to prevent unwanted misretrievals. In ACT-UP, assumptions about context elements for each retrieval are explicitly specified. Thus, ACT-UP currently forgoes some ACT-R constraints:

- *strict harvesting* (automatic buffer clearing and learning as chunks): chunks are learned explicitly
- *all-encompassing spreading activation* (all buffers may spread activation): cues are specified during retrieval in ACT-UP
- *unselective partial matching* (the full retrieval request is matched partially): ACT-UP retrieval distinguishes hard and soft constraints

To see a typical chunk creation, retrieval and learning cycle, suppose the model knows initially, via declarative memory, a fact such as *the lawyer is in the dungeon*:

```
(learn-chunk                                  (add to DM)
  (make-fact :name 'l-d-fact                  (new chunk)
             :person 'lawyer
             :location 'dungeon))
```

We can, at model run-time, retrieve and reinforce this chunk:

```
(let ((fact (retrieve-chunk                   (retrieve)
             '(:location dungeon))))          (constraints)
  (if fact (learn-chunk fact)))               (reinforce)
```

Key memory processes such as base-level learning and decay, cue-based memory retrieval, partial matching and their parametrization are equivalent to ACT-R 6. Also available are associative learning as in ACT-R 5 (Anderson, 1993) and Blending (Wallach & Lebiere, 2003). ACT-UP models may define a chunk type hierarchy, and they may derive data structures from chunk types in an object-oriented fashion.

### Procedural skills

ACT-R defines procedural rules as fine-grained instructions of the form *If the buffers contain certain values, then change their values according to another template*. ACT-UP is situated at a higher level of abstraction. Modelers may specify complex *rules* that define sequences of actions and preconditions, similar to a Lisp function. Production rules are not usually evaluated in parallel, unless the modeler relies on *utility learning* to model effects through reinforcement learning. ACT-R's utility learning boosts the likelihood of successful productions being chosen in cases of ambiguity (multiple productions match). ACT-UP allows models to define rules and explicitly group them in *competition sets*. ACT-UP can chose a rule from a competition set. Rewards are explicitly back-propagated as in ACT-R 6 in order to let a model learn which rules lead to desirable outcomes. Thus, the production rule conflict sets used in ACT-R are made explicit in ACT-UP rather than being represented implicitly through overlap in production conditions. Routinization effects, where retrievals from declarative memory are side-stepped through specialized, acquired rules, can also be modeled in ACT-UP (the analogous ACT-R mechanism is *production compilation*). ACT-UP's rules consume simulation time (50ms by default), even though the precise predictions that fall out of an ACT-R model are lost, where the same cycle time is assumed, but where production rules are tightly constrained. In line with Accountable Modeling, we propose to fit the execution duration (within plausible bounds) to the data as free parameters.

## (1) Validity: The Siegler Model

An ACT-UP model implementing the core of an ACT-R model should result in exactly the same performance results. To test such consistency of ACT-UP and ACT-R 6, we translated several ACT-R 6 models to ACT-UP. Here, we show the *Siegler* model from the ACT-R 6 tutorial. The model explains data by Siegler & Shrager (1984), who found patterns in arithmetic problem-solving in 4-year-olds. In making mistakes when answering addition problems, the children often closely under- or overshot the correct result ($2+3=6$), and their erroneous answers were more frequent and strayed further from the target for problems involving larger numbers. The ACT-R 6 model (following Siegler and Shrager's model) explains these data using a combination of partial matching and base-level activations in memory retrieval of arithmetic facts. Similarity between numbers is proportional to their absolute difference, so that close answers may be retrieved ($2+3=5$). Base-level activations for more frequent addition facts with lower results are higher, leading to more erroneous retrievals and more often for facts involving larger numbers.

The ACT-R 6 model implements a number of deterministic steps: aural presentation, encoding of the numbers, and decoding of the result. The ACT-UP model underspecifies these, as they do not contribute to the variance in the data. The key processing step of the model, the retrieval of arithmetic facts from memory, is accomplished by the following high-level function:

```
(defrule test-fact (arg1 arg2)
  (let ((fact
         (retrieve-chunk                      (retrieve)
           '(:chunk-type plus-fact)           (hard constraints)
           nil                                (no retrieval cues)
           (list :addend1 arg1                (soft
                 :addend2 arg2))))              constraints)
    (if fact (plus-fact-sum fact))))          (extract sum)
```

Model initialization sets base-levels and similarities (in 24 lines of Lisp code), using function calls largely compatible

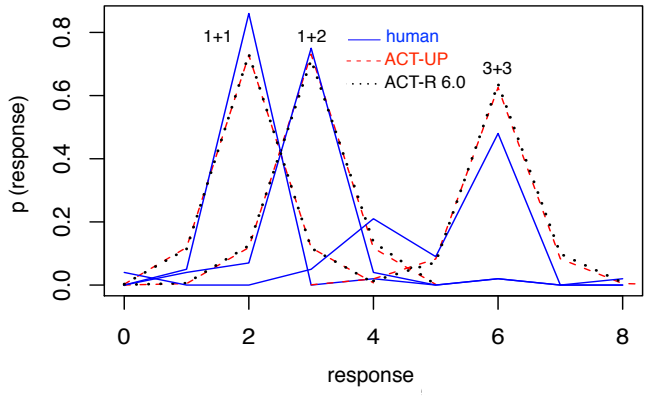


Figure 1: The Siegler and Shrager (1984) data, showing the three distributions of subject's answers to the arithmetic problems 1+1, 1+2 and 3+3, and the simulation results of the model implemented in ACT-R 6 and ACT-UP.

with ACT-R 6. Four architectural parameters are set equally in both variants (retrieval threshold, transient noise, base-level learning (off), and mismatch penalty coefficient).

Both model variants achieved the same correlation (0.966 in ACT-R 6 vs. 0.968 in ACT-UP) and mean deviation (0.053 vs. 0.052) with the data (1000 runs). Figure 1 shows the distribution of the subjects' answers to three of the six problems and demonstrates that the predictions of ACT-UP match closely those of ACT-R 6.

## (2) Scalability: A Model of Language Evolution

A multi-agent model was implemented to reflect the emergence of a domain language common to a group of agents after repeated, goal-oriented interactions (Reitter & Lebiere, 2009). In the *Pictionary* games of the empirical study providing data for this modeling exercise, each participant had to convey given meanings via drawings (without words) to another participant. In this model, a language was defined as a set of concept-representation pairs, where a concept was one of 20 target concepts (e.g., *hospital*), and the representation consisted of three drawings of concrete objects (e.g., *building*, *ambulance*, *syringe*). For novel concepts, the model drew from an ontology (graph with weighted associations) linking concepts to related drawings; relatedness was inferred from co-occurrence information in a large text corpus. Known concept-representation pairs were stored in declarative memory. Prototyping the model in ACT-R 6 proved difficult for several reasons. The model was complex, and possible execution paths through the approximately 40 production rules were not evident. Further, the model needed many iterations to show convergence. Parallelization between eight agents and repeated model execution (without reset) was difficult to achieve for technical reasons. We estimate the expended time to be around two person-months. The prototype's results never approached an acceptable fit with the data on a qualitative or quantitate level.

A functional prototype of the model using an initial version of ACT-UP was developed in less than two weeks with the benefit of a task well understood. We focused on plausibility within the ACT-R theory: No data structures were held in memory beyond what could be stored in a buffer; the domain language used declarative memory as intended. Production rules were abstracted using loops, conditionals and ACT-UP commands, owing to the fact that skill acquisition was not part of the model. The model was split up into several functions which could be individually inspected and tested (e.g., "draw", "recognize"). ACT-UP functions were used to inspect the activation of target chunks at retrieval times (base-level, spreading activation) and export those to be visualized along a time-line. With this model, we were able to establish good qualitative empirical correspondence with data from experiments that compared a small community of eight participants interacting in changing pairs, to a set of participants interacting in four one-on-one dyads.

Recently, the model scaled well to multi-agent simulations with 1000 agents and 84 million game interactions (two stateful agents, one concept per game) in about 36 CPU hours. Further work is planned to evaluate scalability to memory-intensive long-term tasks.

## (3) Efficiency: A Sentence Production Model

The third case study involves a model that was implemented in both ACT-R 6 and in ACT-UP. It involves a model of sentence production (Reitter, 2008), focusing on the syntactic process, and explaining syntactic priming data that show that subjects are more likely to choose one syntactic variant over another if that variant was presented as a prime ("The girl gave the dog a bone" vs. "The girl gave a bone to the dog."). The model begins with a simple semantic representation (Verb: <give>, Agent: <girl>, Theme: <bone>, Goal: <dog>). Beginning with the verb, it chooses words and their syntactic forms describing how those words can combine (the verb has two forms, yielding the two variants above). Base-level activation of the appropriate syntactic chunks held in DM and spreading activation from the meaning as described above determine which form of sentence is produced. Base-level learning and associative learning (in ACT-UP only) lead to a range of priming effects.

The ACT-R 6 model consists of 30 productions, 7 chunk types, and a variable number of chunks that are created for each word and for several syntactic forms. The ACT-R 6 production rules resulted in 720 lines of code. Base-level activations and associations between words and syntactic forms are initialized programmatically from a corpus of spoken, transcribed and syntactically parsed English. The model was evaluated according to its qualitative and quantitative predictions of syntactic priming effects using a small number of sample sentences. As in the empirical data, syntactic priming depends on the frequency of syntactic constructions and the distance between target sentence and prime.

Studies also show that syntactic priming is much increased when lexical material in the sentence is repeated between prime and target. The model postulated that this was due to learning of associations between lexical or semantic and syntactic chunks–a suggestion that was tested empirically in terms of its theoretical predictions, but associative learning

was not available to the ACT-R 6 model. Consequently, the sentence production model and various initialization and simulation functions were formulated in ACT-UP over the course of about 3 eight-hour workdays. The core function encoding procedural knowledge (sequences of retrievals, conditionals, a loop) has 82 lines of code.[2] The resulting model explained the data including the lexical repetition effect.

In head-to-head comparison, the ACT-R 6 model runs at a speed of 14 sentences per second. The ACT-UP variant produces 380 sentences per second, despite being more specific than necessary to explain the data.[3] This purely technical speed-up translates to a substantial advantage for the modeler: not only is the debugging and experimentation cycle considerably faster, but larger models of more realistic tasks can be run in larger multi-agent simulations, thereby significantly extending the applicability of cognitive models.

## (4): Extensibility and Rapid Prototyping: the Dynamic Stocks&Flows Model

The fourth model is another case of rapid prototyping. It illustrates how we could quickly implement a well-documented approach to graded decision-making. *Instance-based learning* (IBL, Gonzalez et al., 2003) stores episodes encoding past decisions and their observed performance in declarative memory. Retrieval then blends those episodes together, weighing their recency and frequency in line with ACT-R's *base-level learning* and *partial matching*.

In an entry (Reitter, 2010) to the *Dynamic Stocks&Flows* modeling challenge[4], IBL was used in two ways. The task in this challenge had subjects extrapolate the change in a given quantity from previous observations. Change rates could be steady (the quantity following a linear function) or harder to predict, including non-linear changes or discontinuous and noisy sequences. IBL modeled the participant's estimates of the change rate and the future value of the quantity. Careful analysis of empirical data showed an interesting pattern: variability often suddenly decreased after about 20 iterations of the task; depending on subject and change function, variability could be grouped into very low and higher pools: subjects were highly precise, or not precise at all. This led to the second use of IBL: a metacognitive layer, which allowed the model to monitor its performance at the task and choose from one of several strategies. Some of these strategies led to precise estimates of the quantity through mental arithmetic, and other strategies used IBL, as described above, to make an educated guess.

The model used declarative memory for its core transaction: declarative chunks store the quantity estimates and the performance monitoring episodes. Blending of stored episodes is implemented (and available) at the ACT-UP level. One free parameter in the model specified the duration of calculations and the wait time between iterations (an underspecified model component); we fitted the parameter from available subject data. The results were plausible given the experimental design. Other parameters were held at their ACT-R defaults; blending parameters were optimized. The model won the challenge by best predicting transfer performance to a set of unknown conditions, indicating that accountable modeling has the potential of increasing generalization of models by focusing on the key processes underlying performance. The same model was later run in an extensive parameter exploration exercise, in which selected architectural and model parameters were systematically varied, with millions of model runs on a computing cluster (Gluck et al., 2010). The exploration included a manipulation that switched individual strategies on and off.

The DSF model exemplifies Accountable Modeling through the decision to not describe the visual and motor interaction with the experiment. While a portion of the data might have been explained by the subject's use of the graphical user interface, neither timing, eye-tracking or mouse movement data were available for validation. Thus, the model underspecifies motor and sensor components.

## (5) Reusability: Lemonade Game Agent

The final test case illustrates the re-use of model components. We used a cognitive model in ACT-UP to explore the performance of metacognition in a multi-agent game competition[5]. The DSF challenge model (Reitter et al., 2010) provided the metacognitive layer choosing one of multiple strategies. The model plays a location game (*Lemonade Stand*), where the optimal choice of strategy depends on the strategies played by the two opponents. We designed a metacognitive model that chooses from a wide range of elementary prediction and action strategies based on their track record. The metacognitive model always outperforms all single-strategy models we implemented in a round-robin tournament. The metacognitive layer only had to be minimally adapted: the core functions for learning and blending retrieval were identical; only the task-specific objective functions were redefined. ACT-UP suggests useful compartmentalization: its functions take a set of arguments and return a value; they are intended to be side-effect free apart, of course, from changes to the state of the model. As a consequence, they are reusable in new contexts.

The Lemonade Game agent is not a classical cognitive model, explaining existing empirical data. Instead, its metacognitive layer generates predictions. Not all individual strategies are formulated fully within the theory; thus, we demonstrate a way to combine cognitive and purely algorithmic models. Difficulties arose when the model was readied for submission to a competition, which required Java: in such cases, we got the best use of ACT-UP as a prototyping tool, but had to re-implement the model once validated.

[2] The relatively faithful translation means that we do not fully follow *Accountable Modeling*: the model is overly specific.

[3] Both models keep a similar-size declarative memory, require similar retrievals; ACT-UP adds associative learning. Both models were run in the same LISP environment, without debug output. ACT-R 6 tracing and logging were off, decision tree building on. Optimized learning for ACT-UP and ACT-R 6 at 3 chunks.

[4] `www.hss.cmu.edu/departments/sds/ddmlab/modeldsf/`

[5] `tech.groups.yahoo.com/group/lemonadegame/`

## Discussion

Most importantly, we want to propose a modeling paradigm that institutionalizes what is often already the case whenever cognitive models depend on the combination of just a few, specific properties of the architecture. A series of case studies provided the basis for our introduction to ACT-UP. We demonstrated the use of ACT-UP in high-fidelity models with up to 1000 parallel agents; we showed cases of rapid prototyping and of the re-use of model components. Quantitative predictions of ACT-UP parallel those of ACT-R.

The models are intended to integrate within one *architecture*. The emergence of more complex, perhaps unexpected behavior then follows from the reuse and combination of models that describe behavior in much more complex, perhaps even realistic environments. ACT-UP models are intended to be underspecified where data cannot account for the specific claims encoded by the model. Such a modeling paradigm appears not only sensible (as it is evidence-based): it also supports scaling up modeling efforts and extending them to new applications. Architectural flexibility is gained through liberal combination of components, not unlike what was proposed by Cassimatis (2002).

Are such models still models of *cognitive* processes, or are they merely computer programs? First, the execution directives (Lisp clauses) specify the model at a higher level than do production rules: both can be seen as computer programs. Importantly, production rules can implement any algorithm, and could, thus, be derived from the ACT-UP model. Thus, ACT-UP models are not theoretically more powerful. Second, temporary storage of variables and even complex data structures enables the modeler to write implausible ACT-UP models, just like large buffers provide a way to exceed what is cognitively believable. Plausibility is not guaranteed unless modeler discretion is entirely removed, which has not been accomplished under any implementation of the theory. Third, ACT-UP's and ACT-R's longer-term storage model (chiefly declarative memory) is an example of strong constraints on what a modeler can do in these formalisms, as opposed to a non-cognitively motivated program.

## Conclusion

We see the current state of ACT-UP as an experimental step to scale up cognitive modeling and extend its areas of applicability. Much work remains to be done. Perceptual and motor components are not yet completed, and parallelism as in ACT-R as well as in its multitasking variant Salvucci et al. (2009) is desirable. The combination of pattern recognition algorithms with ACT-UP may provide for a plausible implementation of the IF part of production rules, possibly to automatically bootstrap and optimize models from sample runs. Larger-scale, long-term simulations will show the limits of the architecture. Still, the wide variety of test cases presented demonstrates scalability w.r.t. modeling effort and computations, and has taken a step towards the integration of high-fidelity cognitive models in complex cognitive systems.

## Acknowledgements

The authors acknowledge funding for this work from the Air Force Office of Scientific Research (MURI 7 - FA95500810356).

## References

Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* Oxford, UK: Oxford University Press.

Bothell, D. (2005). *ACT-R 6.0 Reference Manual.* Retrieved 12/2009, from `act-r.psy.cmu.edu/actr6/reference-manual.pdf`

Cassimatis, N. L. (2002). *Polyscheme: A cognitive architecture for integrating multiple representation and inference schemes*. Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Gluck, K. A., Stanley, C. T., L. Richard Moore, J., Reitter, D., & Halbrügge, M. (2010). Exploration for understanding in model comparisons. *Journal of Artificial General Intelligence (to appear)*.

Gonzalez, C., Lerch, F., & Lebiere, C. (2003). Instance-based learning in dynamic decision making. *Cognitive Science*, *27*, 591-635.

Kirchner, W. K. (1958). Age differences in short-term retention of rapidly changing information. *Journal of Experimental Psychology*, *55*(4), 352-358.

Laird, J. E., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1-64.

Newell, A. (1973). You can't play 20 questions with nature and win. In W. Chase (Ed.), *Visual information processing.* New York, N.Y.: Academic Press.

Reitter, D. (2008). *Context effects in language production: Models of syntactic priming in dialogue corpora*. Unpublished doctoral dissertation, University of Edinburgh.

Reitter, D. (2010). Metacognition and multiple strategies in a cognitive model of online control. *Journal of Artificial General Intelligence (to appear)*.

Reitter, D., Juvina, I., Stocco, A., & Lebiere, C. (2010). Resistance is futile: Winning lemonade market share through metacognitive reasoning in a three-agent cooperative game. In *Proceedings of the 19th Behavior Representation in Modeling & Simulation (BRIMS).* Charleston, SC.

Reitter, D., & Lebiere, C. (2009). Towards explaining the evolution of domain languages with cognitive simulation. In *Proceedings of the 9th International Conference on Cognitive Modeling (ICCM).* Manchester, UK.

Salvucci, D. D., & Lee, F. J. (2003). Simple cognitive modeling in a complex cognitive architecture. In *Chi '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 265–272). New York, NY, USA: ACM.

Salvucci, D. D., Taatgen, N. A., & Borst, J. P. (2009). Toward a unified theory of the multitasking continuum: from concurrent performance to task switching, interruption, and resumption. In *Chi '09: Proceedings of the 27th International Conference on Human Factors in computing systems* (pp. 1819–1828). New York, NY, USA: ACM.

Siegler, R. S., & Shrager, J. (1984). Strategy choices in addition and subtraction: How do children know what to do? In C. Sophian (Ed.), *The origins of cognitive skills* (p. 229-293). Hillsdale, NJ: Erlbaum.

Stewart, T. C., & West, R. L. (2007). Deconstructing and reconstructing ACT-R: Exploring the architectural space. *Cognitive Systems Research*, *8*(3), 227-236.

Wallach, D., & Lebiere, C. (2003). Conscious and unconscious knowledge: Mapping to the symbolic and subsymbolic levels of a hybrid architecture. In L. Jimenez (Ed.), *Attention and implicit learning.* Amsterdam, Netherlands: John Benjamins.