Cognitive Modeling

Lecture 3: Basic Features of Cogent

Sharon Goldwater

School of Informatics
University of Edinburgh
sgwater@inf.ed.ac.uk

January 18, 2010

Based on the Cogent tutorial held by Rick Cooper at the Cognitive Science Conference, Philadelphia, 2000.

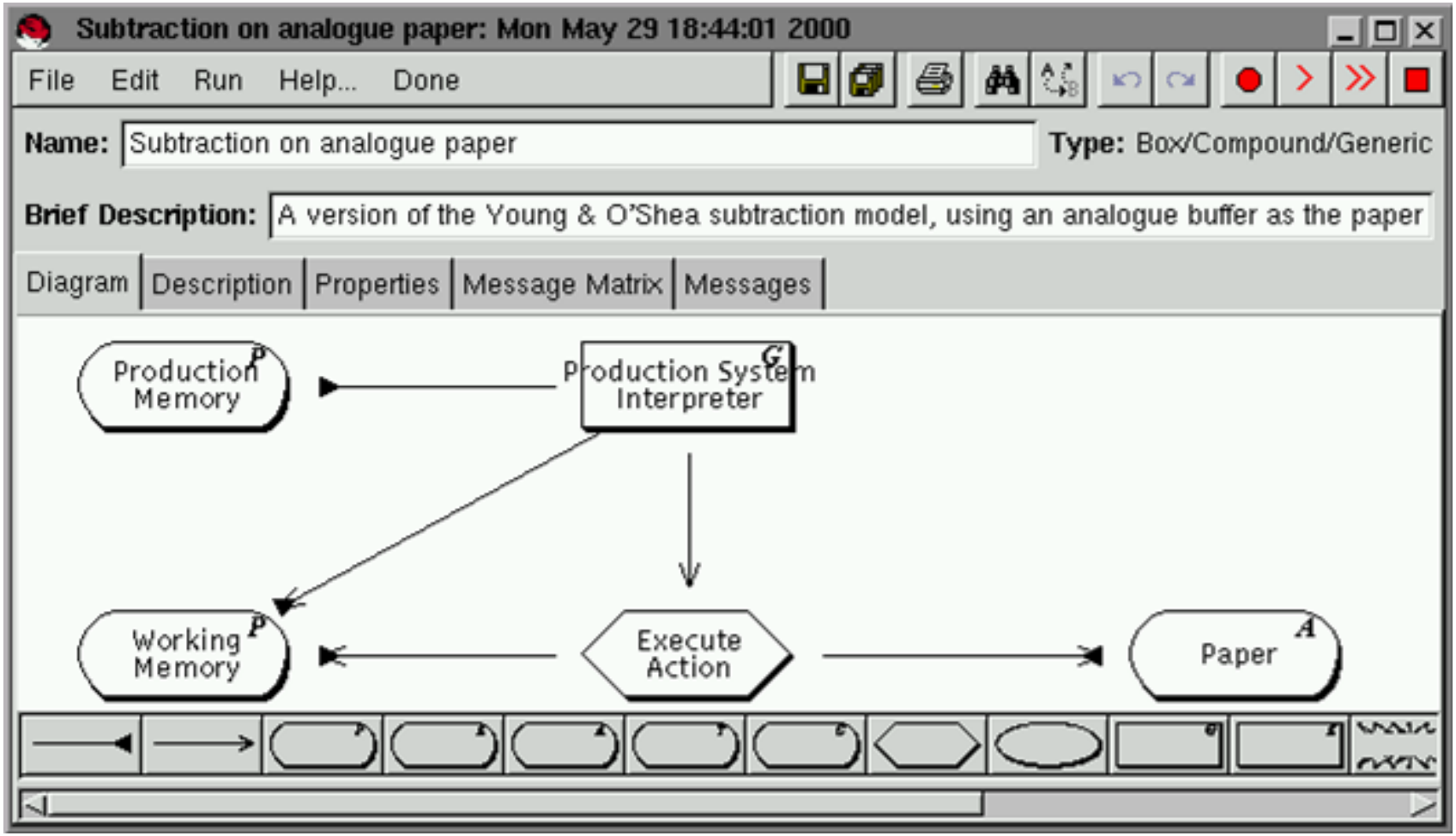
Reading: Cooper (2002: Ch. 2).

Cogent: Principal Features

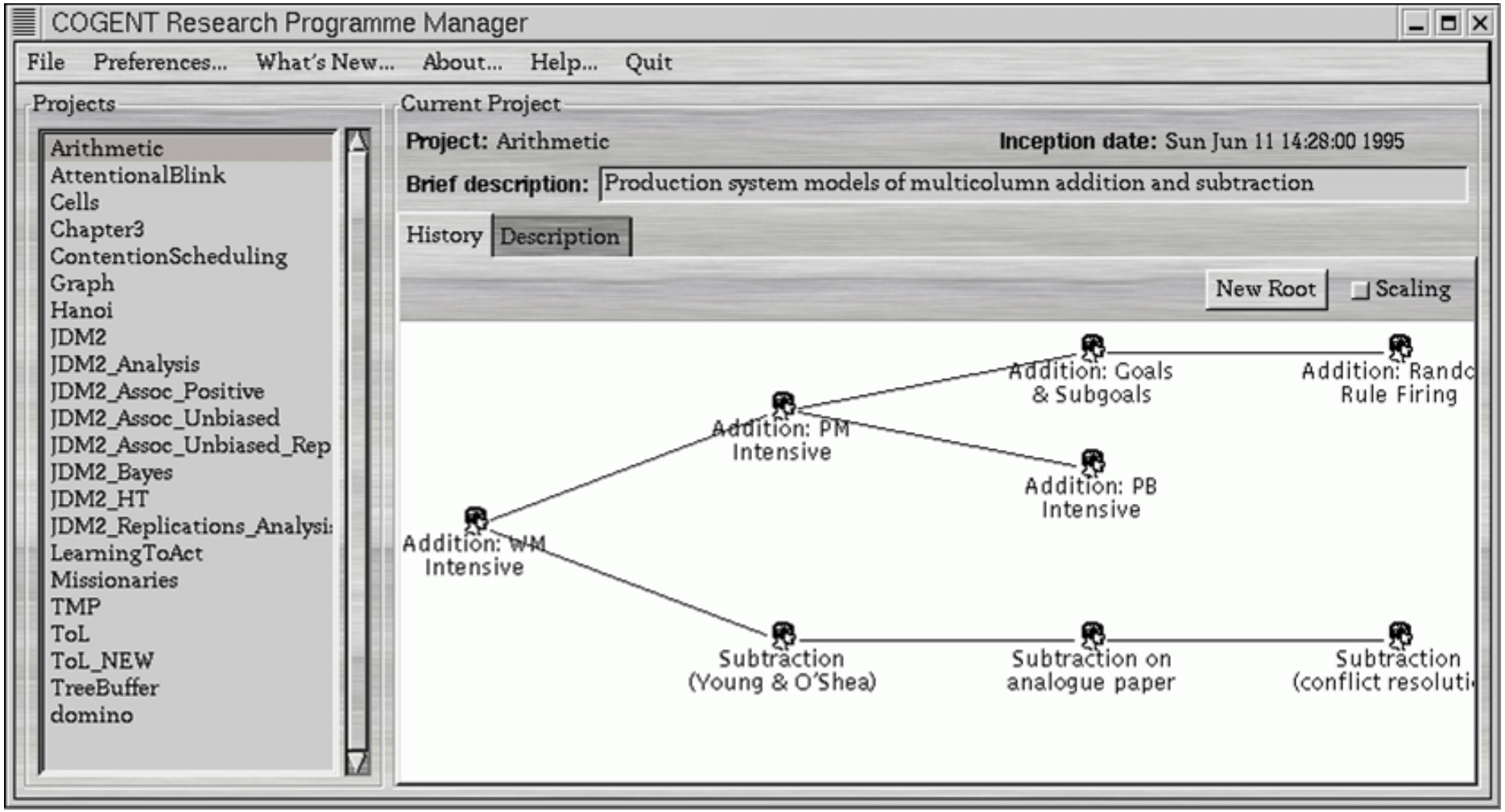
Cogent offers the following features:

- a visual programming environment;
- research program management tools;
- a range of standard functional components;
- an expressive rule-based modeling language and implementation system;
- automated data visualization tools;
- a powerful model testing environment.

Visual Programming in Cogent



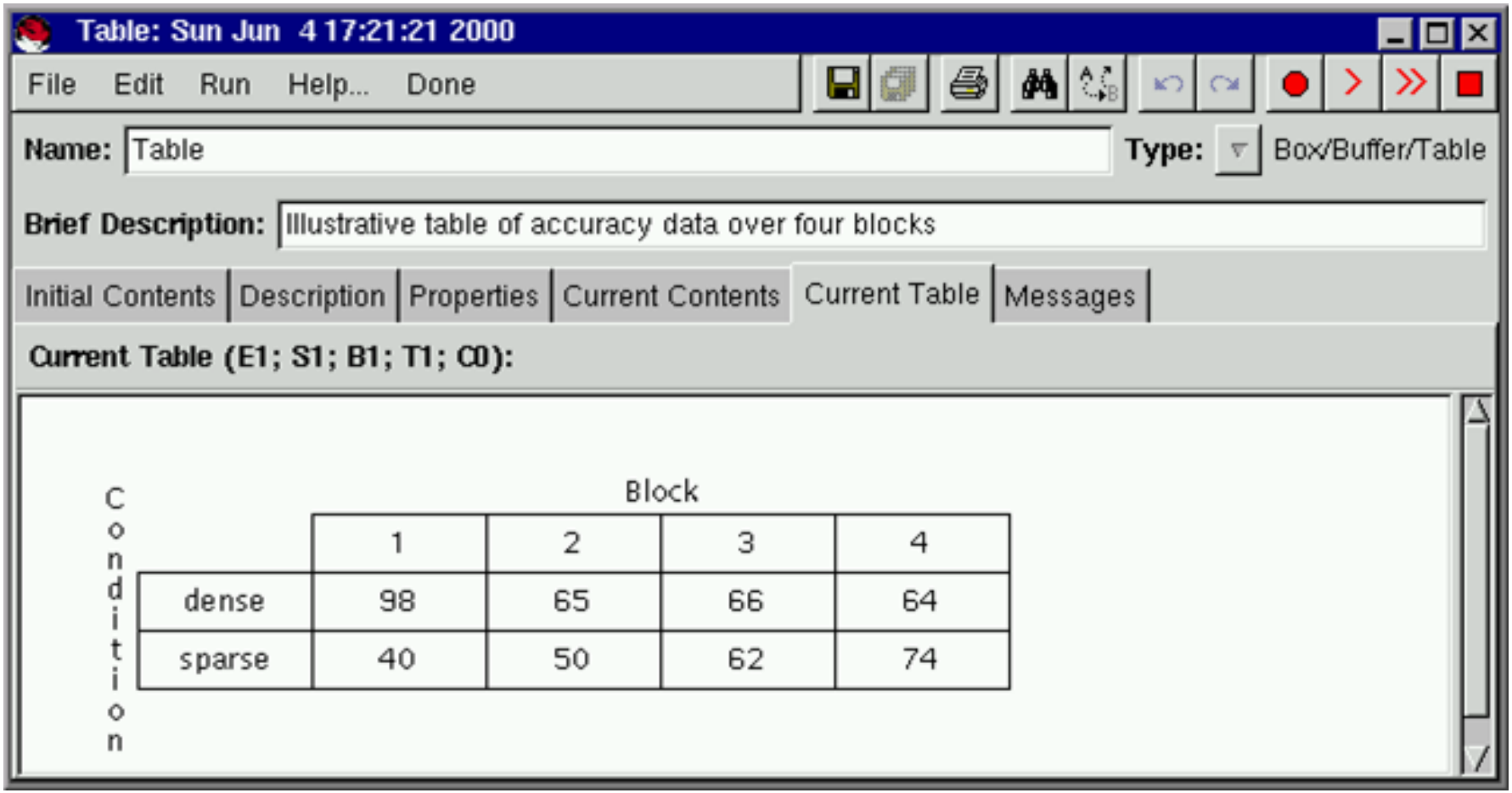
Research Program Management



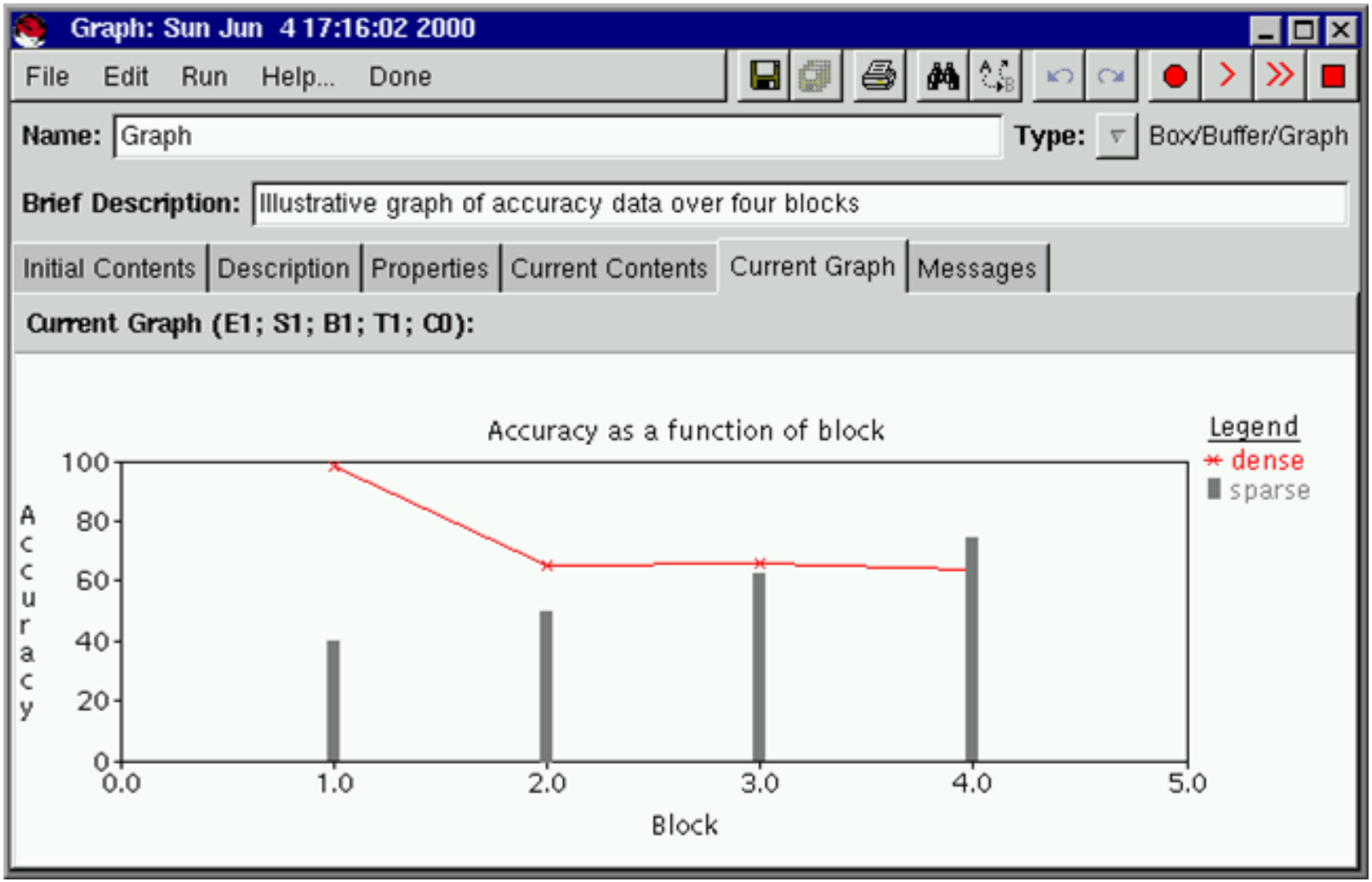
Standard Functional Components

- A library of components is supplied:
  - memory buffers;
  - rule-based processes;
  - simple connectionist networks;
  - data input/output devices.
- Components can be configured for different applications.

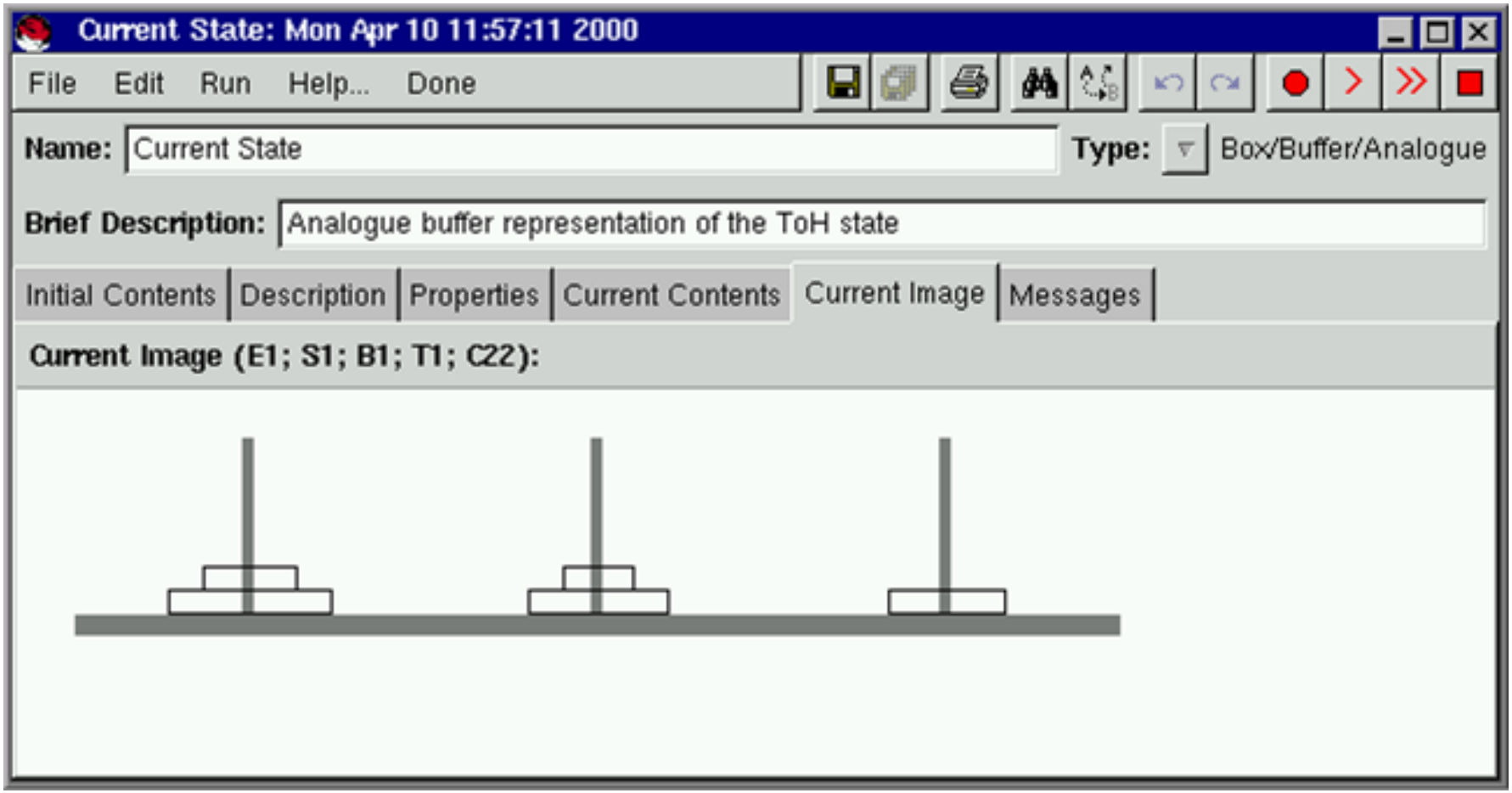
Data Visualization Tools: Tables



Data Visualization Tools: Graphs



Data Visualization Tools: Pictures



The Model Testing Environment

- Visualization tools are dynamically updated;
- facilities are included to trace inter-component communication;
- a flexible "scripting" environment allows:
  - models to be run over multiple blocks of trials;
  - multiple "subjects" to be run over multiple blocks;
  - automated parameter varying "meta-experiments".

Rule-Based Modeling Language

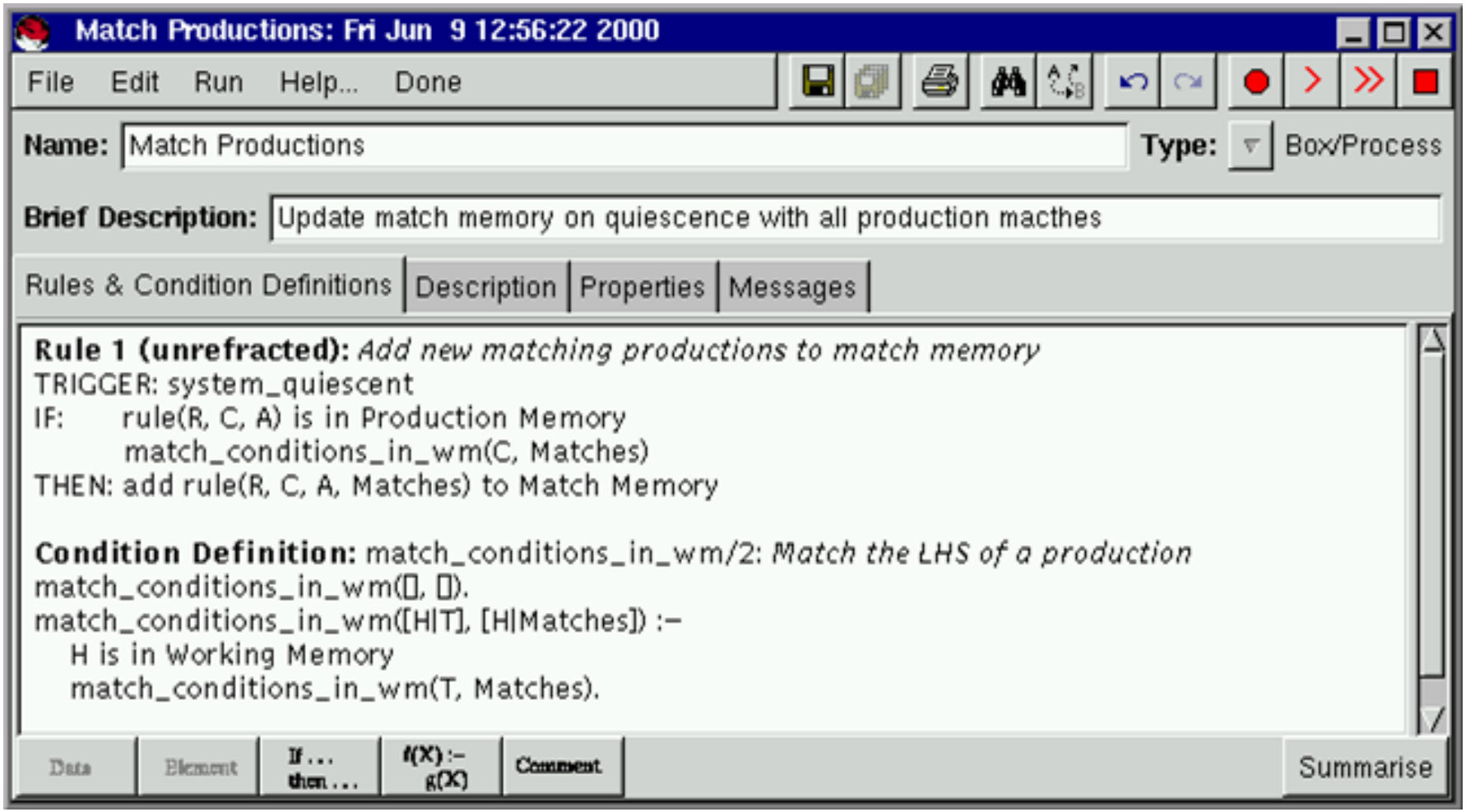
Processes may contain rules such as:

IF `minuend(X)` is in **Working Memory**
   `subtrahend(X)` is in **Working Memory**
THEN add `equal(minuend, subtrahend)` to **Working Memory**
   send `difference(0)` to **Write Answer**

Cogent's representation language is based on **Prolog.**

We will use `teletype` for terms and **boldface** for buffers.

Rule-Based Modeling Language



Basic Syntax

Cogent's representational unit is the *term*. Terms include

- Numbers: reals or integers. Ex: `6`, `6.0`.
- Atoms: any string of letters, digits, or '_' beginning with a lower-case letter. Ex: `apple`, `b0`, `myName`, `response_count`.
- Variables: any string of letters, digits, or '_' beginning with an upper-case letter or '_'. Ex: `Apple`, `B0`, `MyName`, `_count`.
- Lists: for representing sequences; consist of comma-separated terms. Ex: `[a, b, c]`, `[X]`, `[]`.
- Compound terms: for representing structured information; consist of a *functor* and *arguments*. Ex: `word(apple)`, `date(6, jan)`.

Unification

The power of Cogent's representational system comes from *unifying* terms, binding variables to values (or to other variables).

IF `word(Word)` is in **Stimuli**
  the current cycle is `Cycle`
THEN send `memorize(Word)` to **Subject**
  delete `word(Word)` from **Stimuli**
  add `presented(Word, Cycle)` to **Stimuli**

Rules of Unification

- Variables match anything.
- Compound terms must have matching functors and arity.
- Lists must have matching lengths.

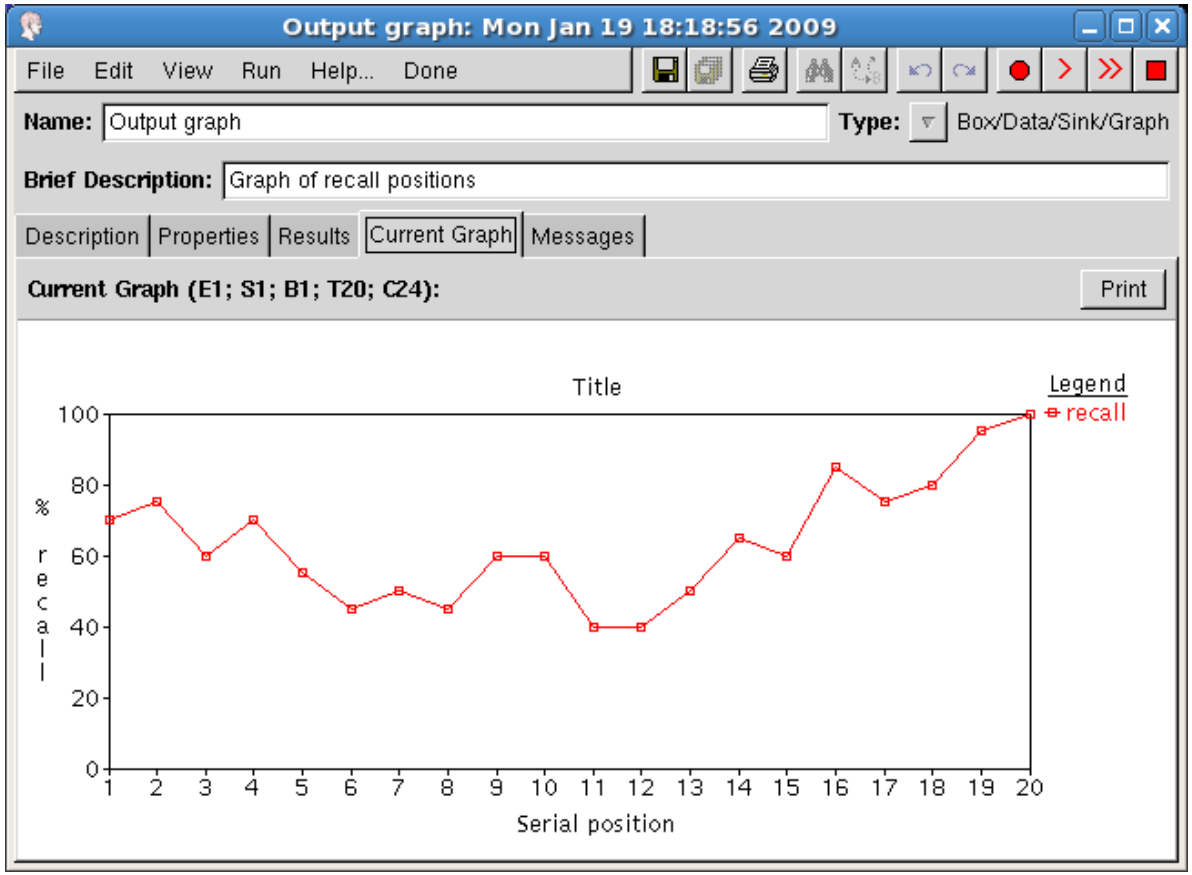
Terms	Unifies as	Bindings
`X, word(Word)`	`word(Word)`	`X` → `word(Word)`
`f(a,B,c), f(X,Y,Z)`	`f(a,B,c)`	`X` → `a`, `Y` → `B`, `Z` → `c`
`f(X), g(a)`	fails	
`f([a,B],[]), f([B,B],B)`	fails	

The Task: Free Recall

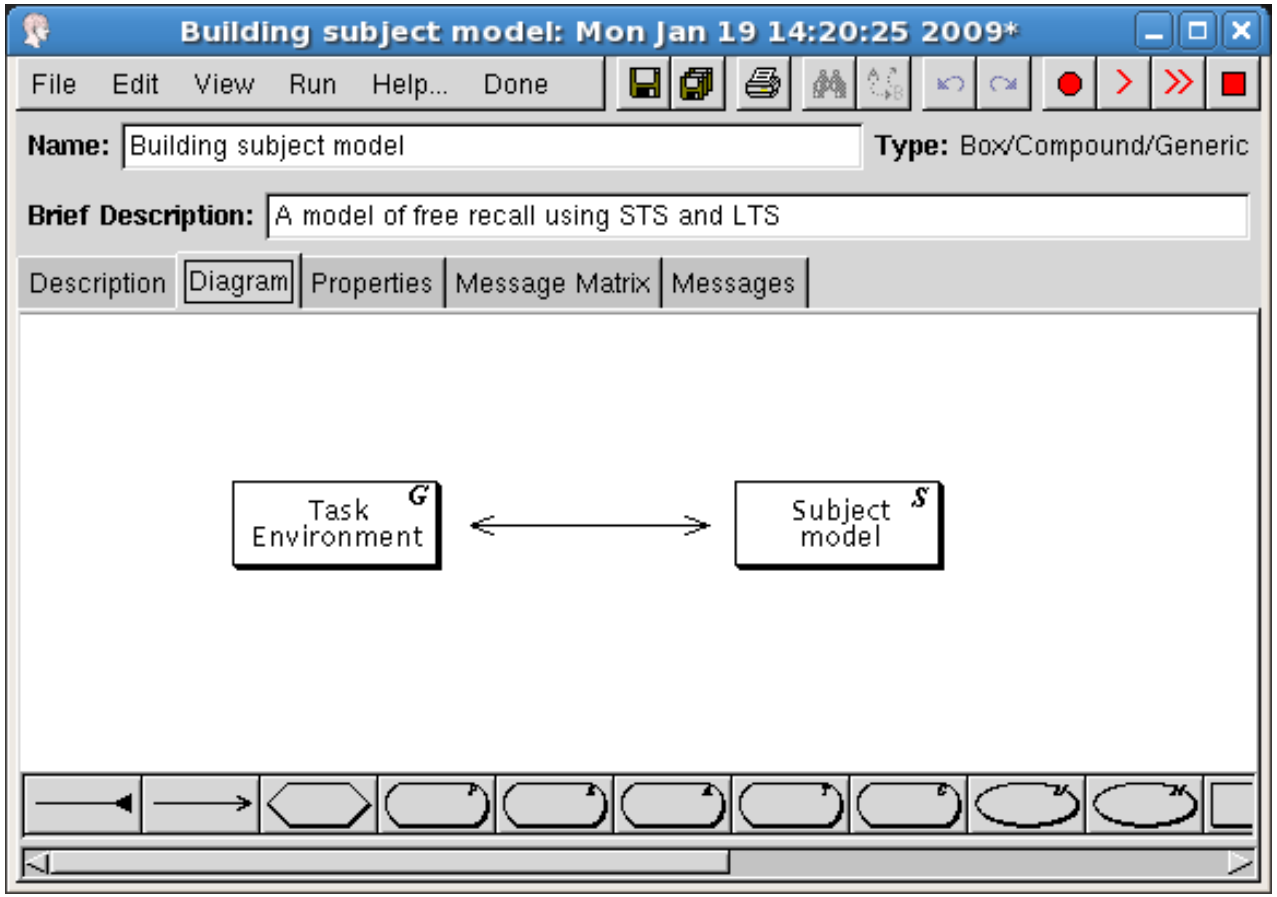
Classical experiment on word learning:

- on each trial, the subject is presented with a list of 25 words;
- the subject is told to try to memorize the words;
- after an interval, the subject must recall as many words as possible.

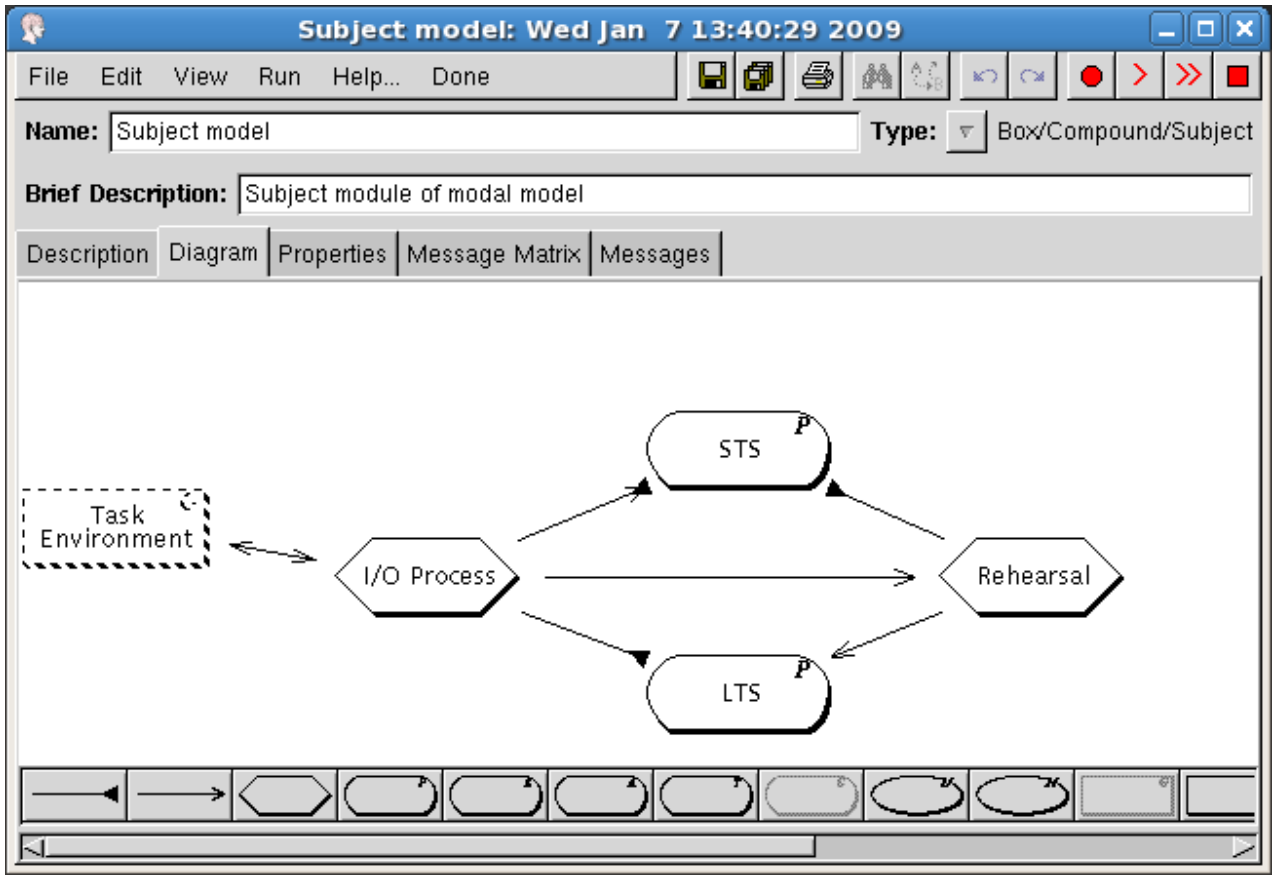
Free Recall: Empirical Findings



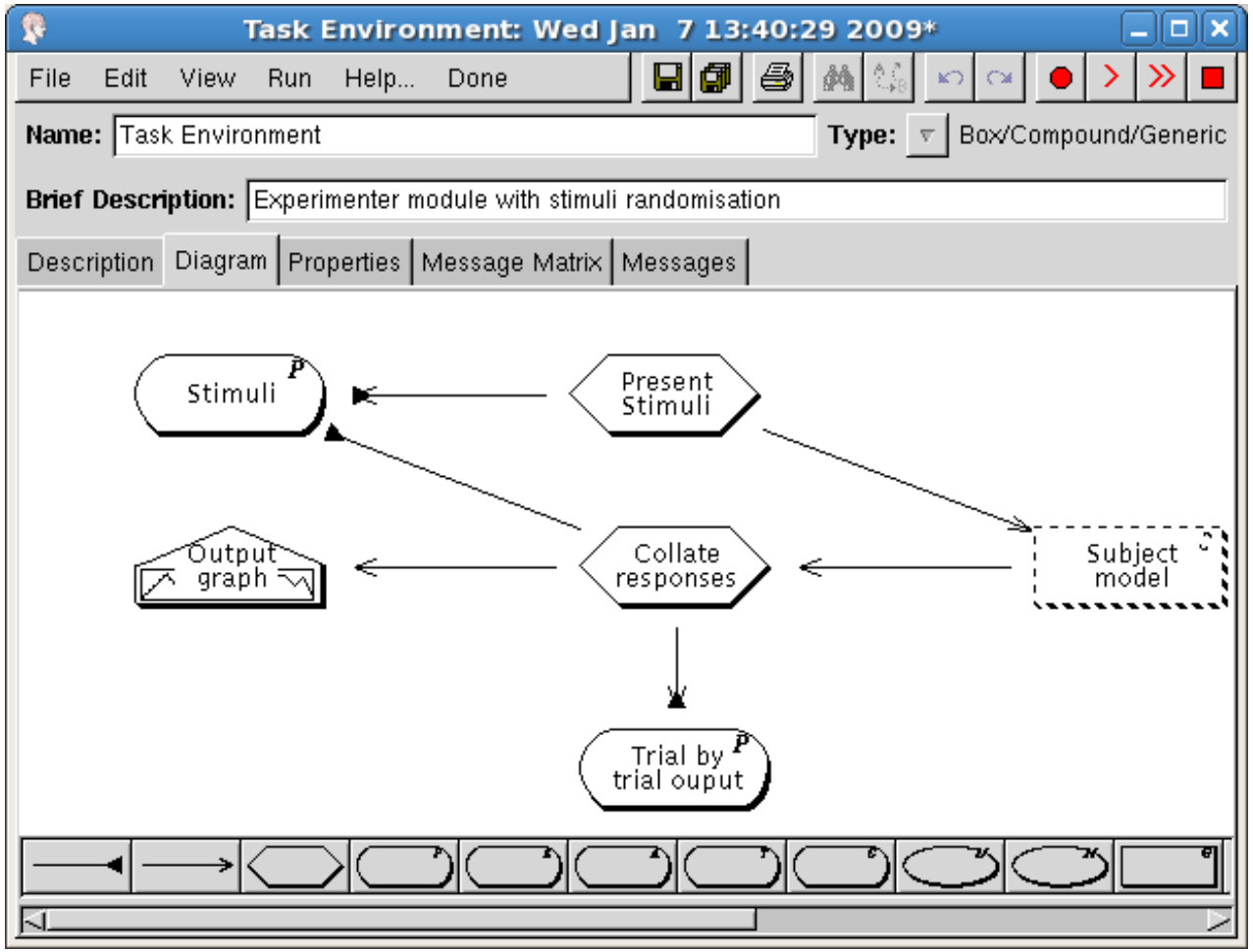
The Modal Model: Top Level



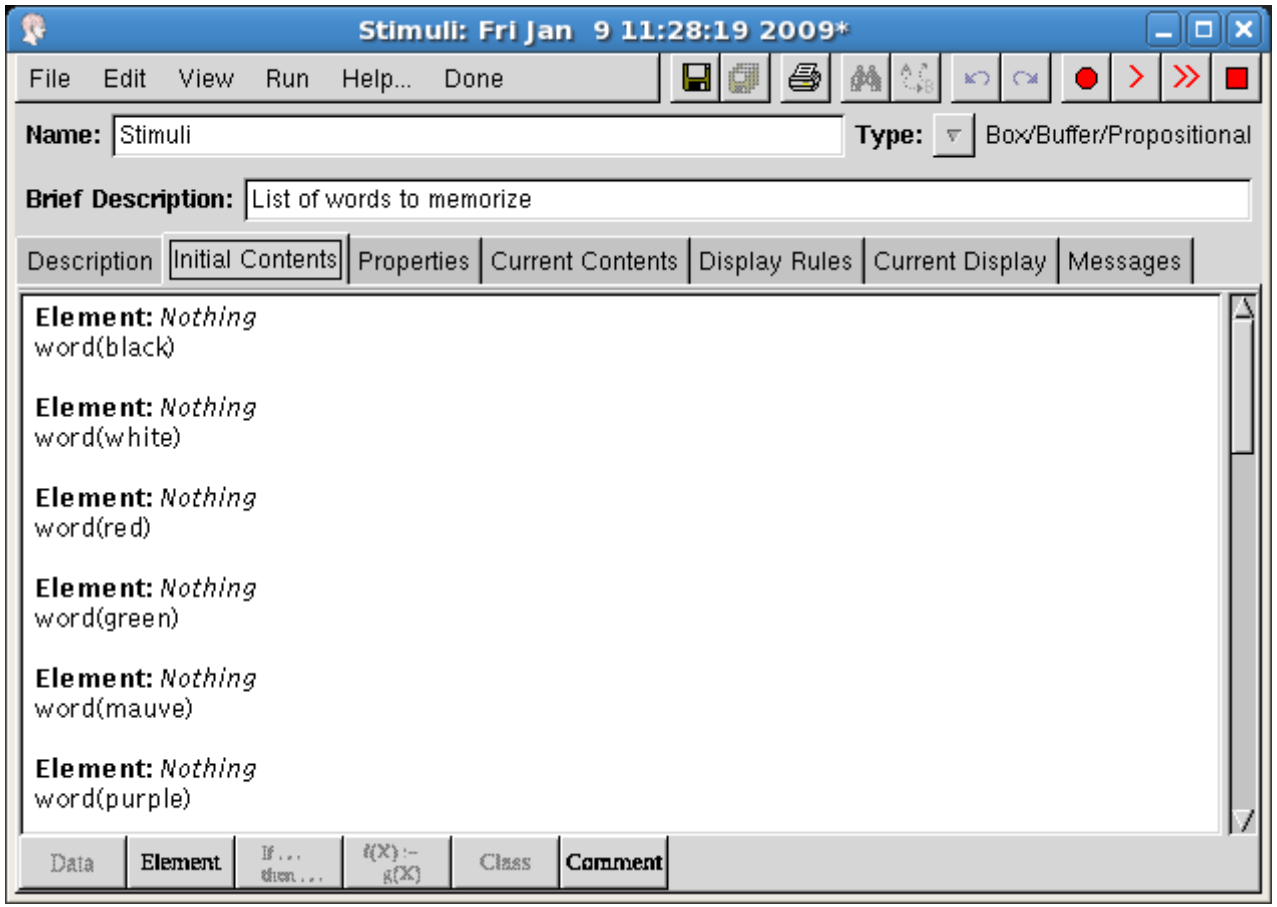
The Modal Model: Subject



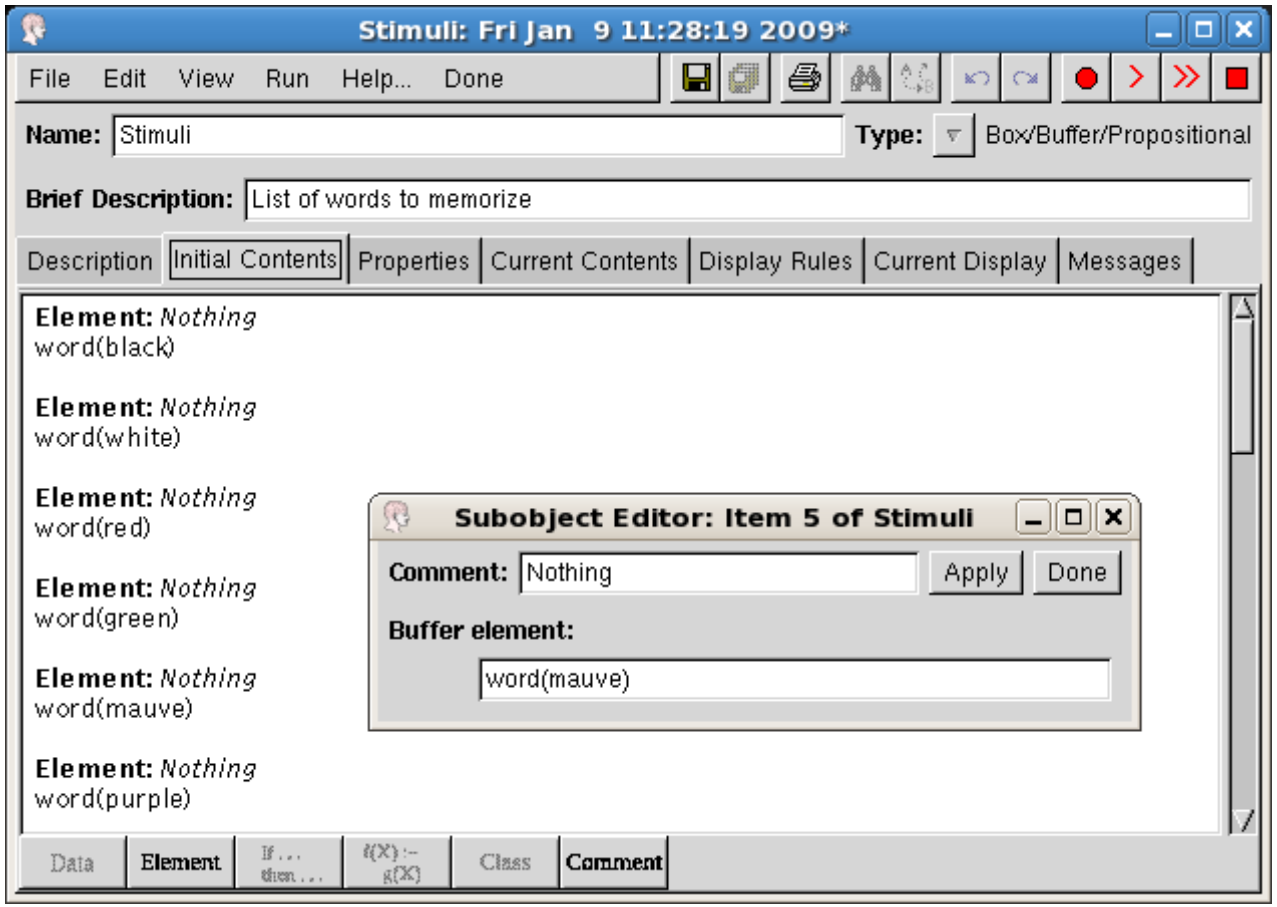
The Modal Model: Task environment



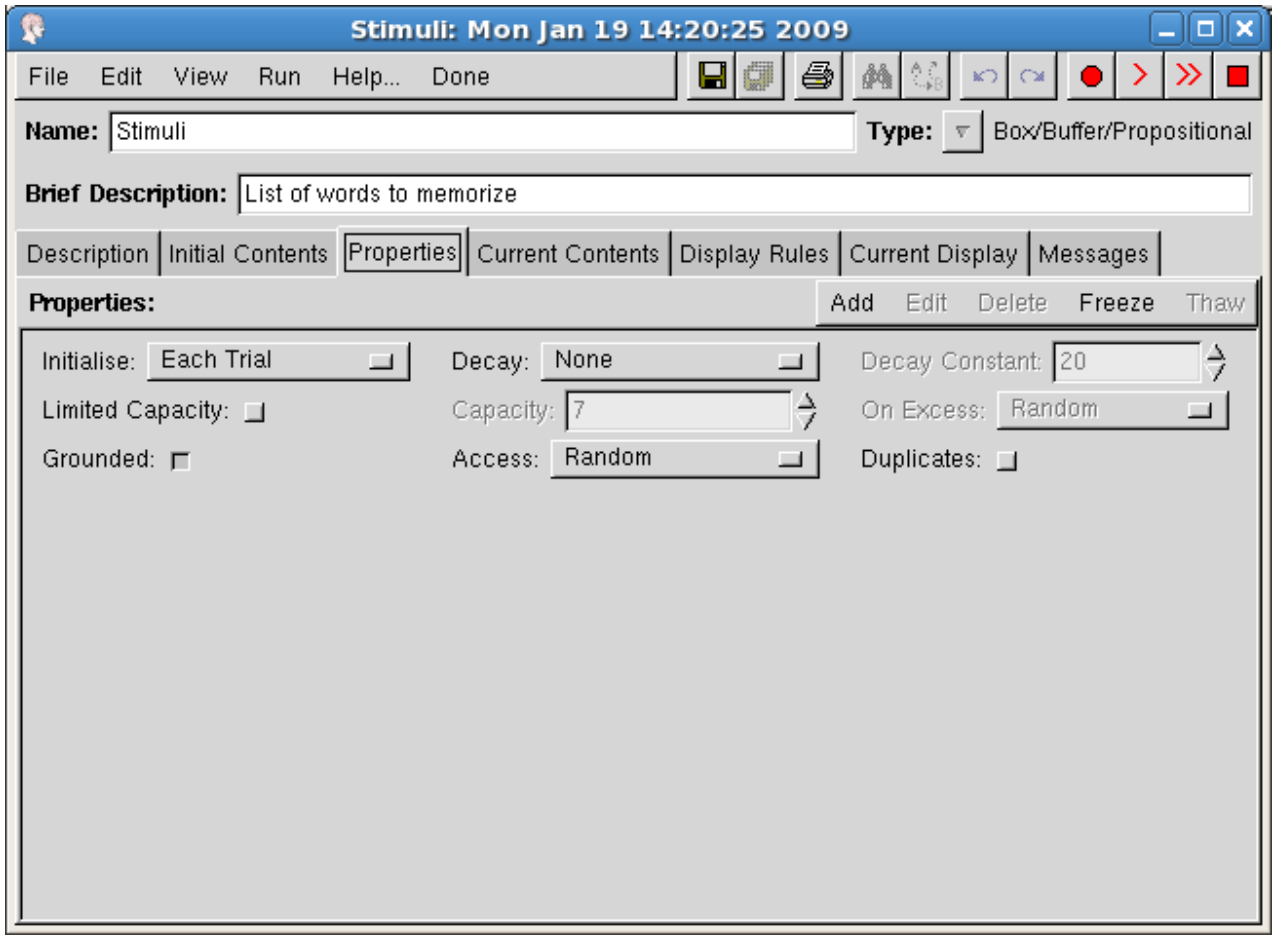
The Modal Model: Stimuli



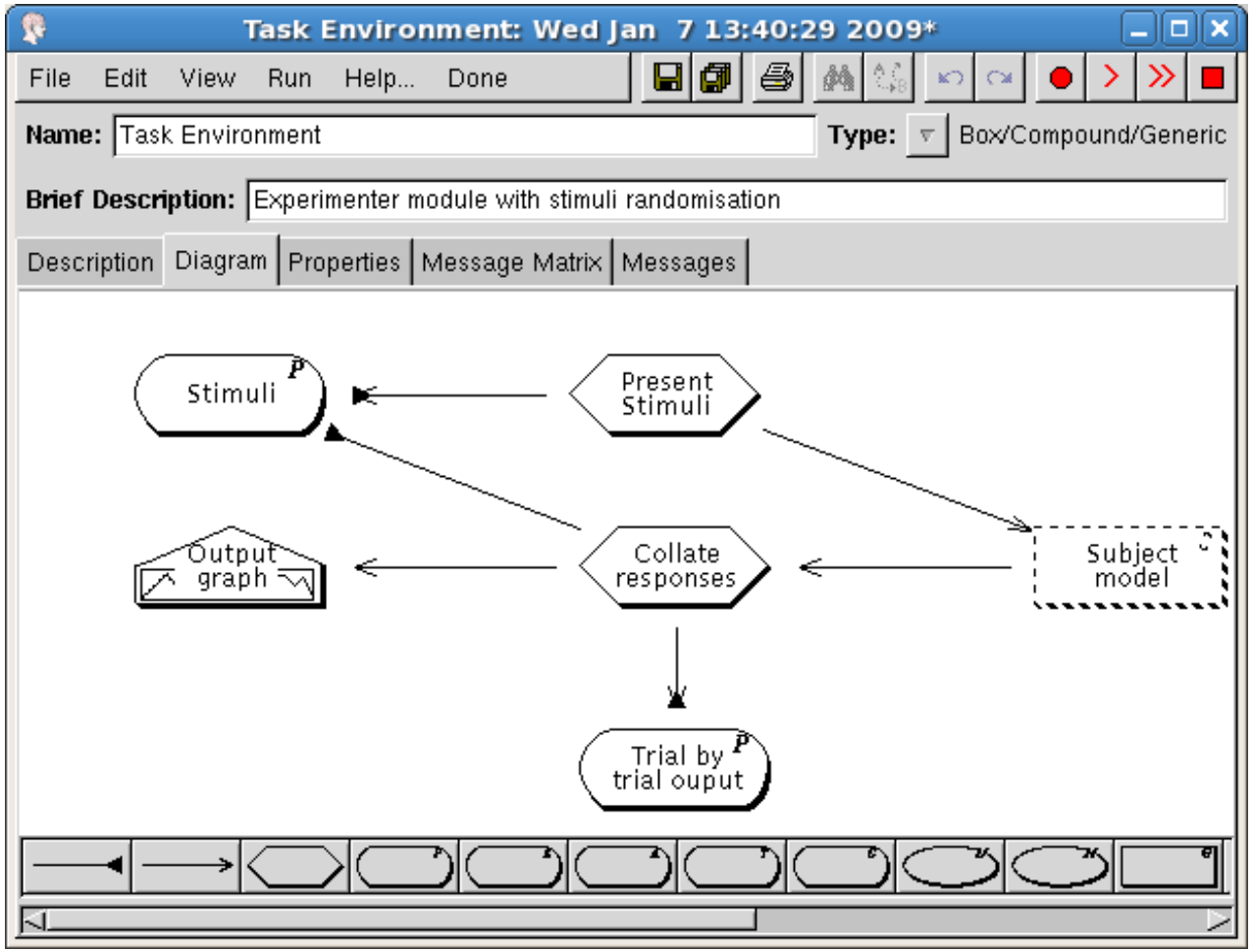
The Modal Model: Editing stimuli



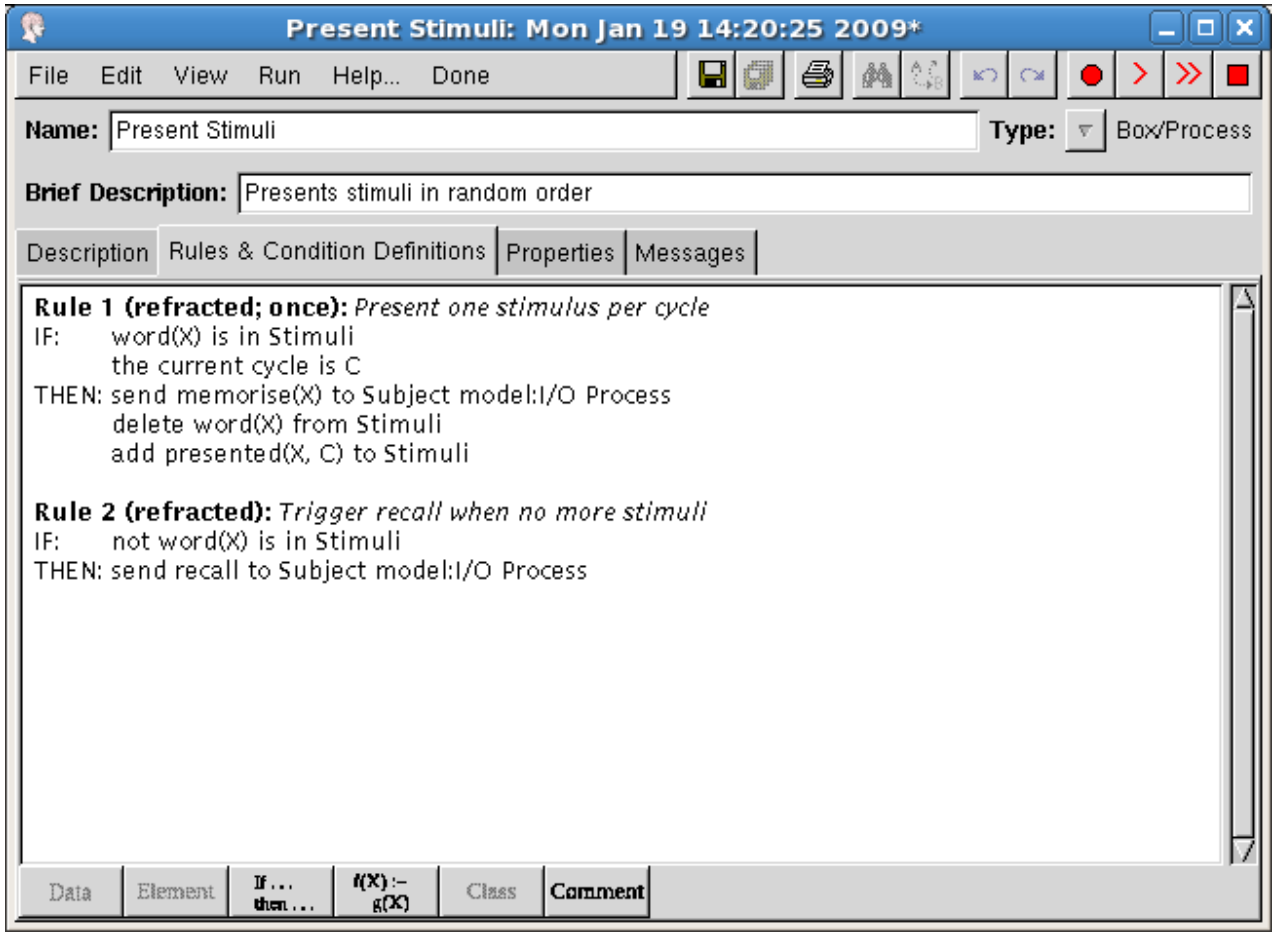
The Modal Model: Stimuli properties



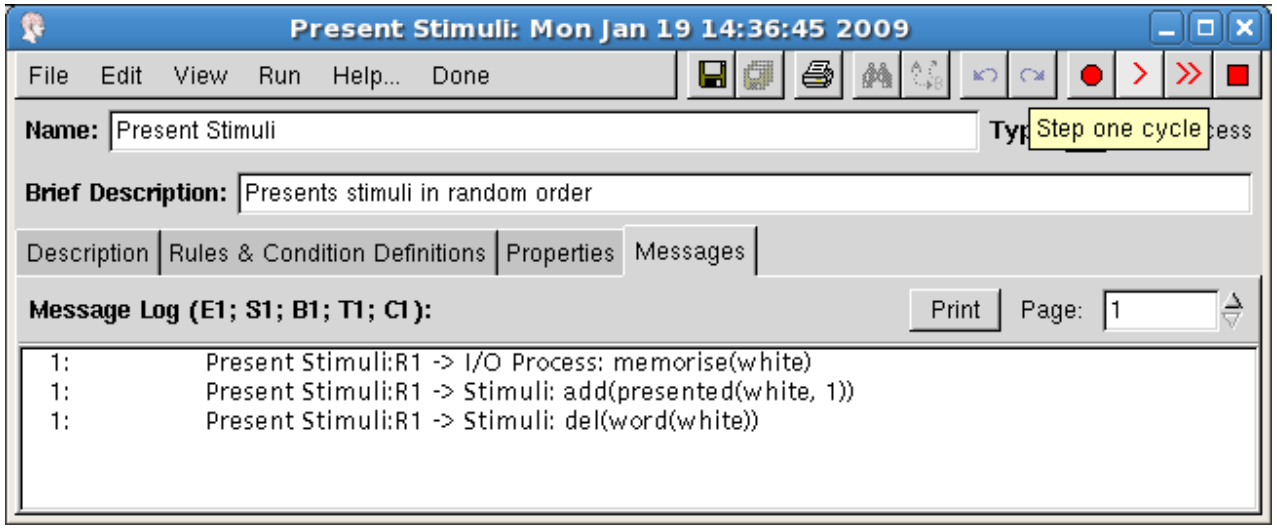
The Modal Model: Task environment

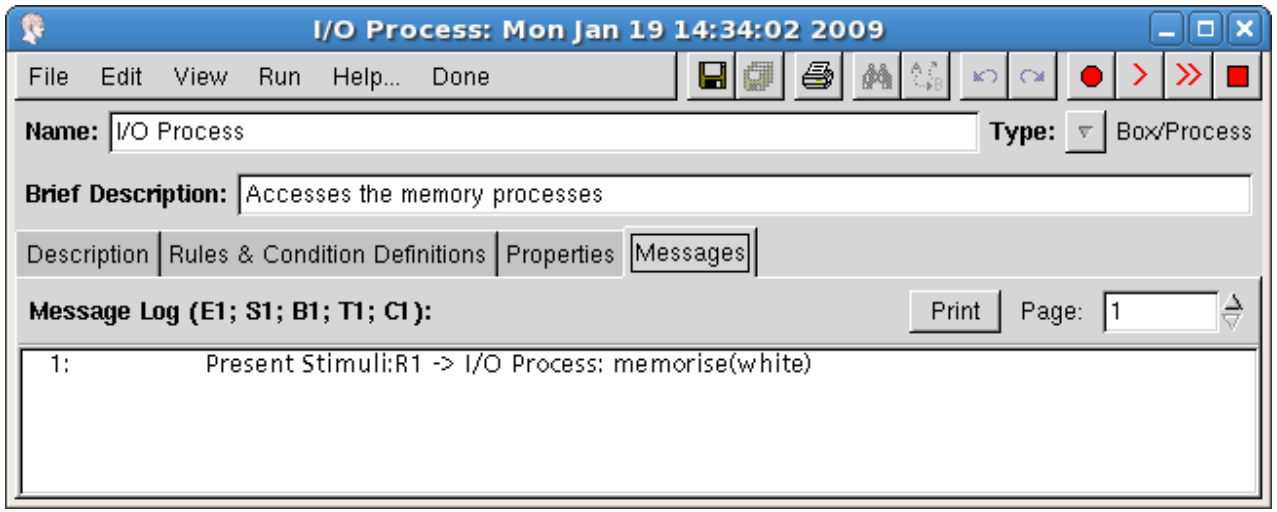


The Modal Model: Presenting stimuli to subject



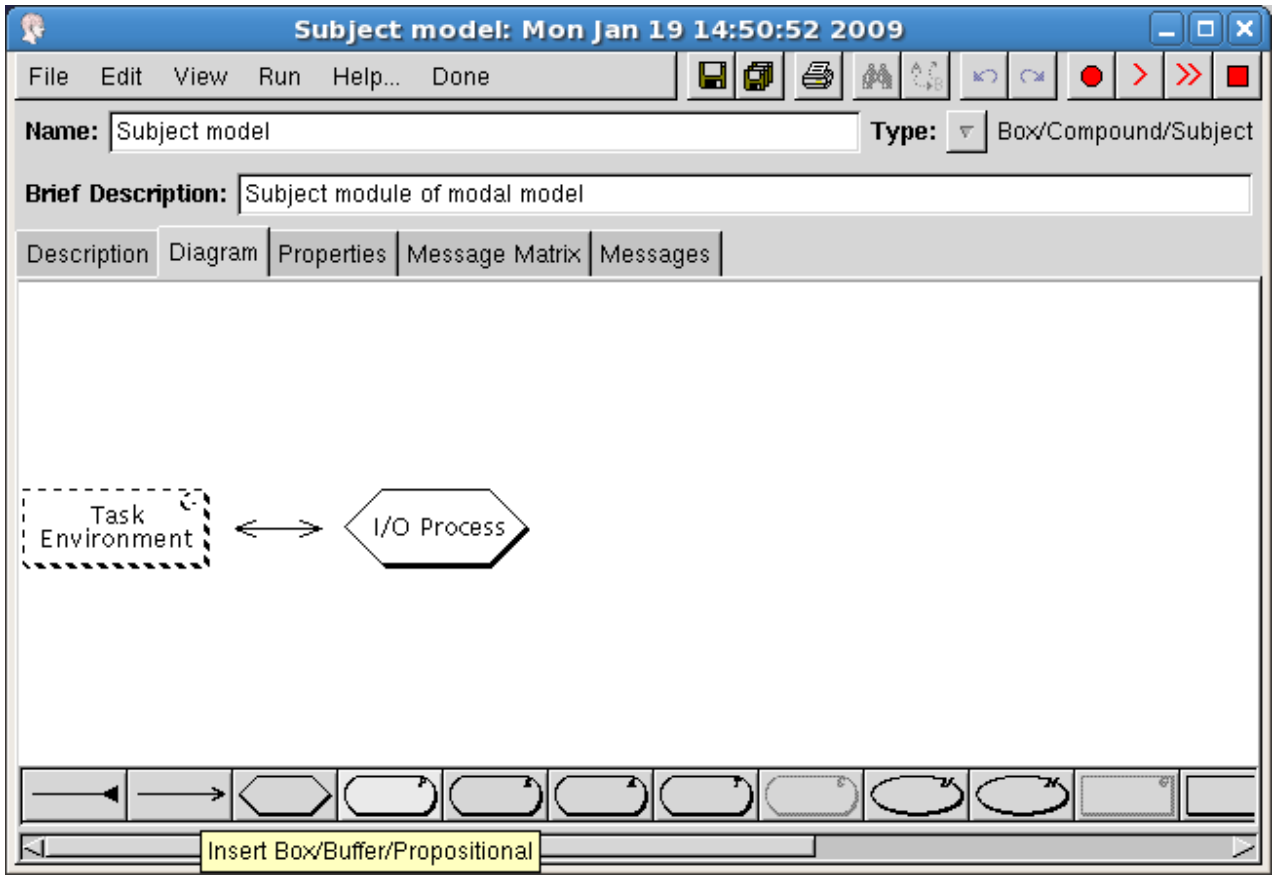
The Modal Model: Messages





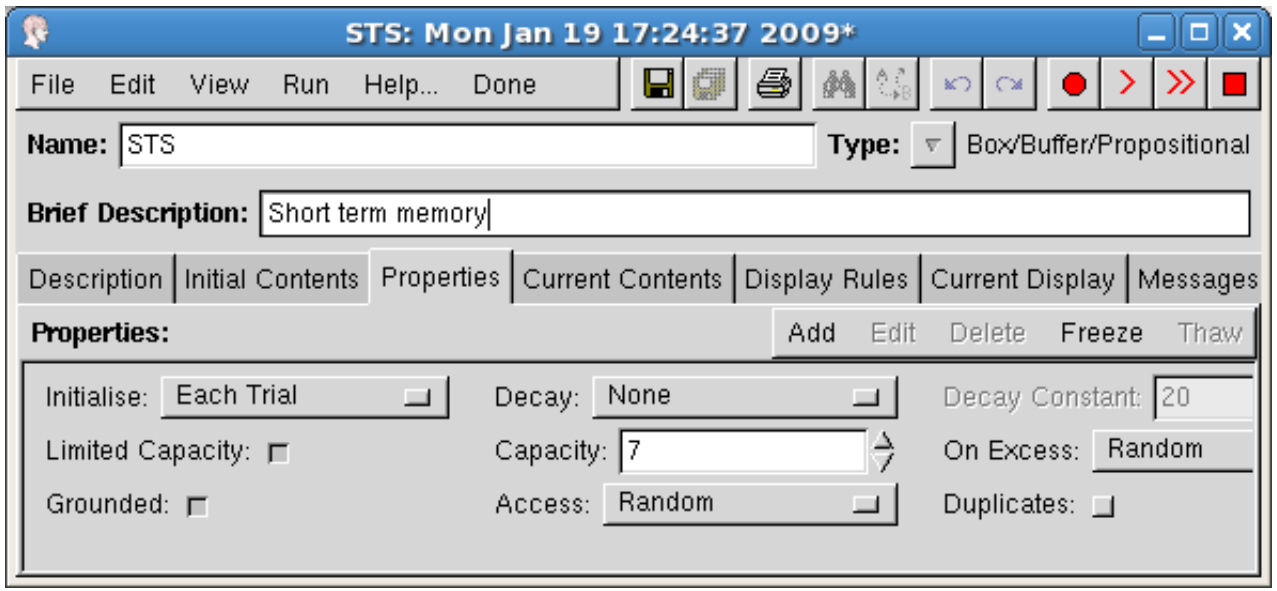
Building the Short Term Store

Create propositional buffer by clicking on button:



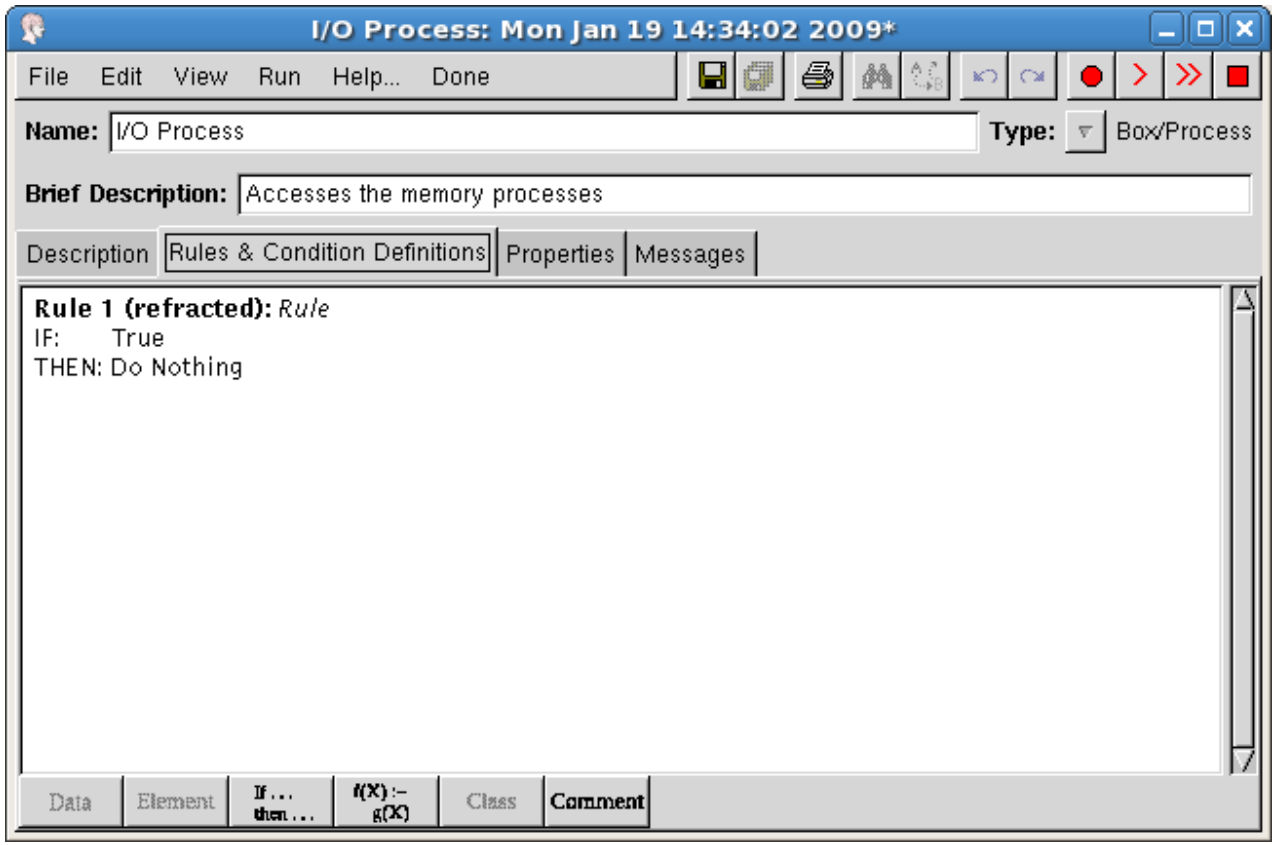
Building the Short Term Store

Double-click new buffer to name it and edit properties. **STS** has Limited Capacity of 7:



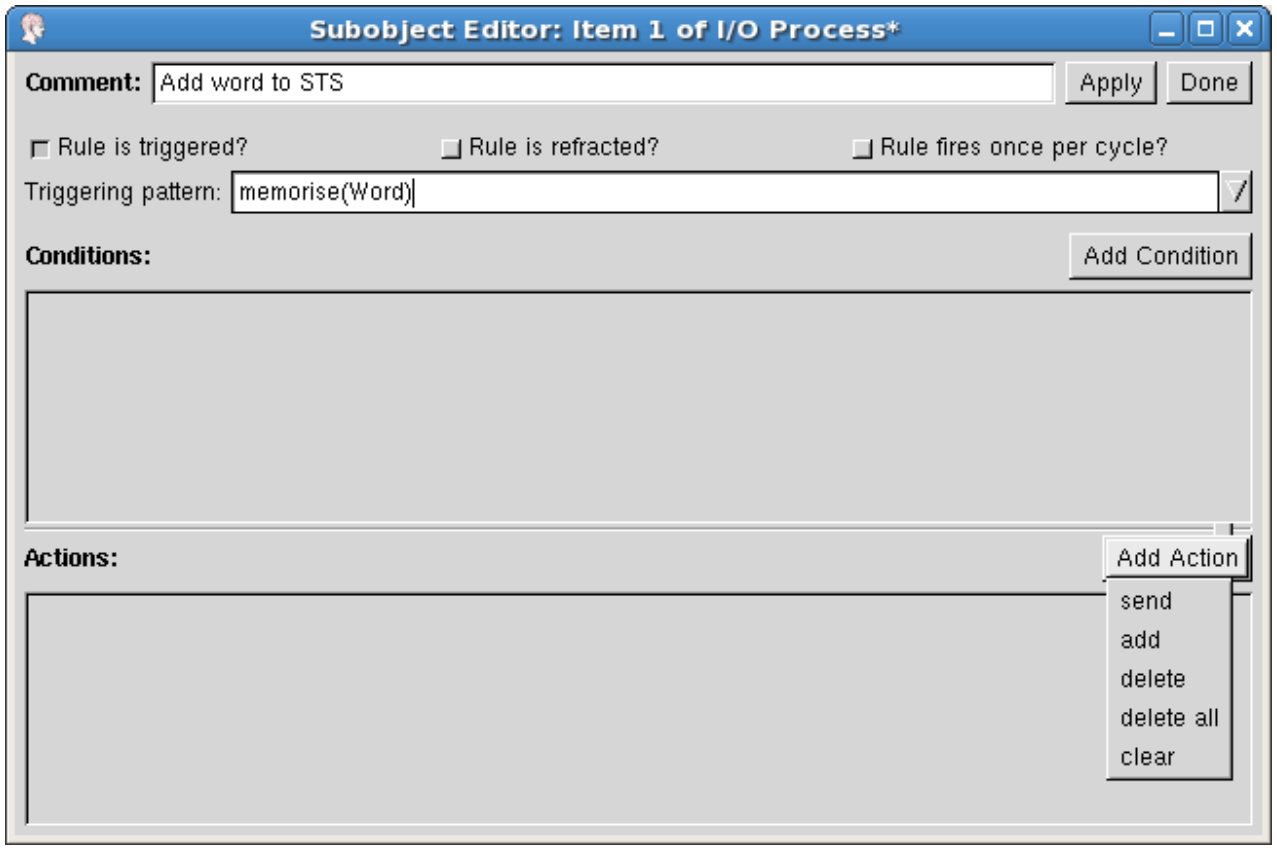
Building the Short Term Store

Open **I/O Process** and add an If...Then... rule:



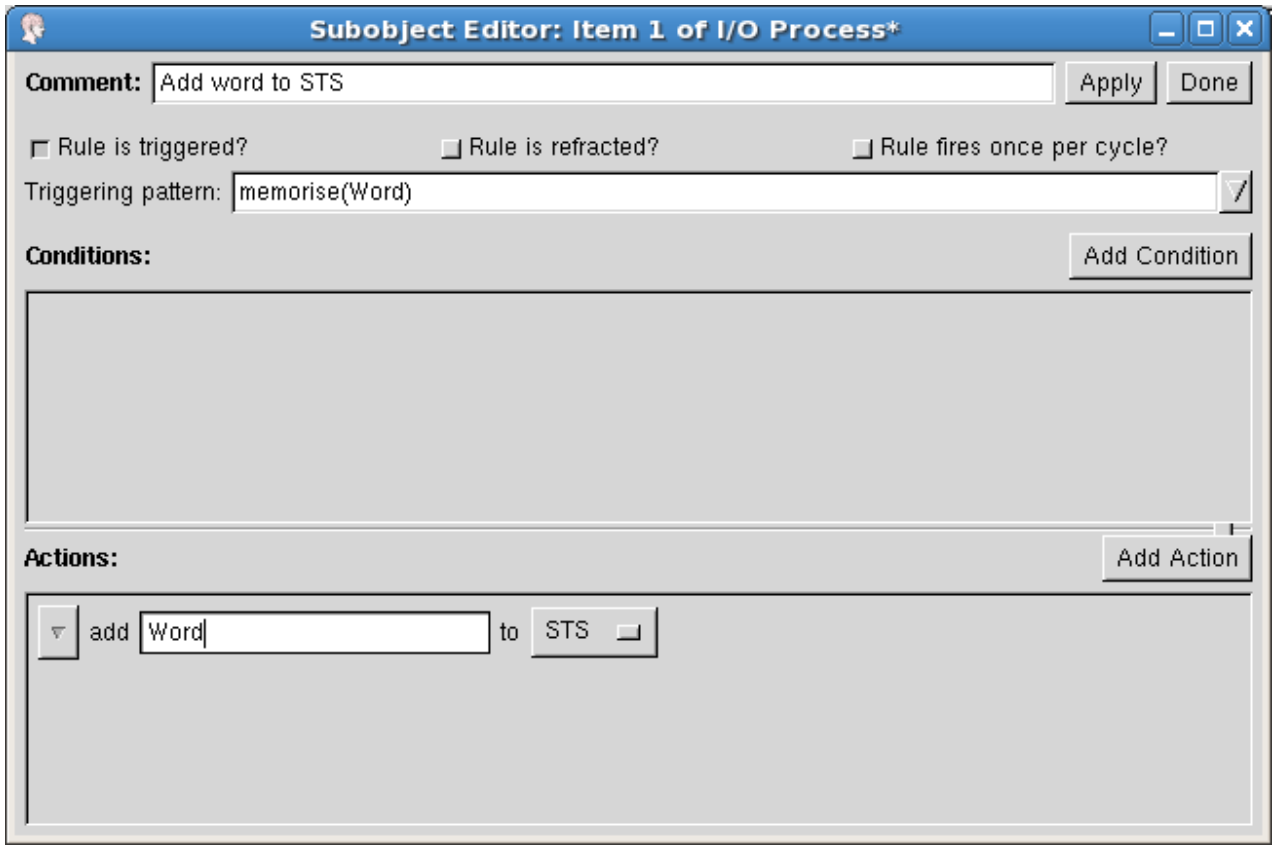
Building the Short Term Store

Double-click rule to edit:



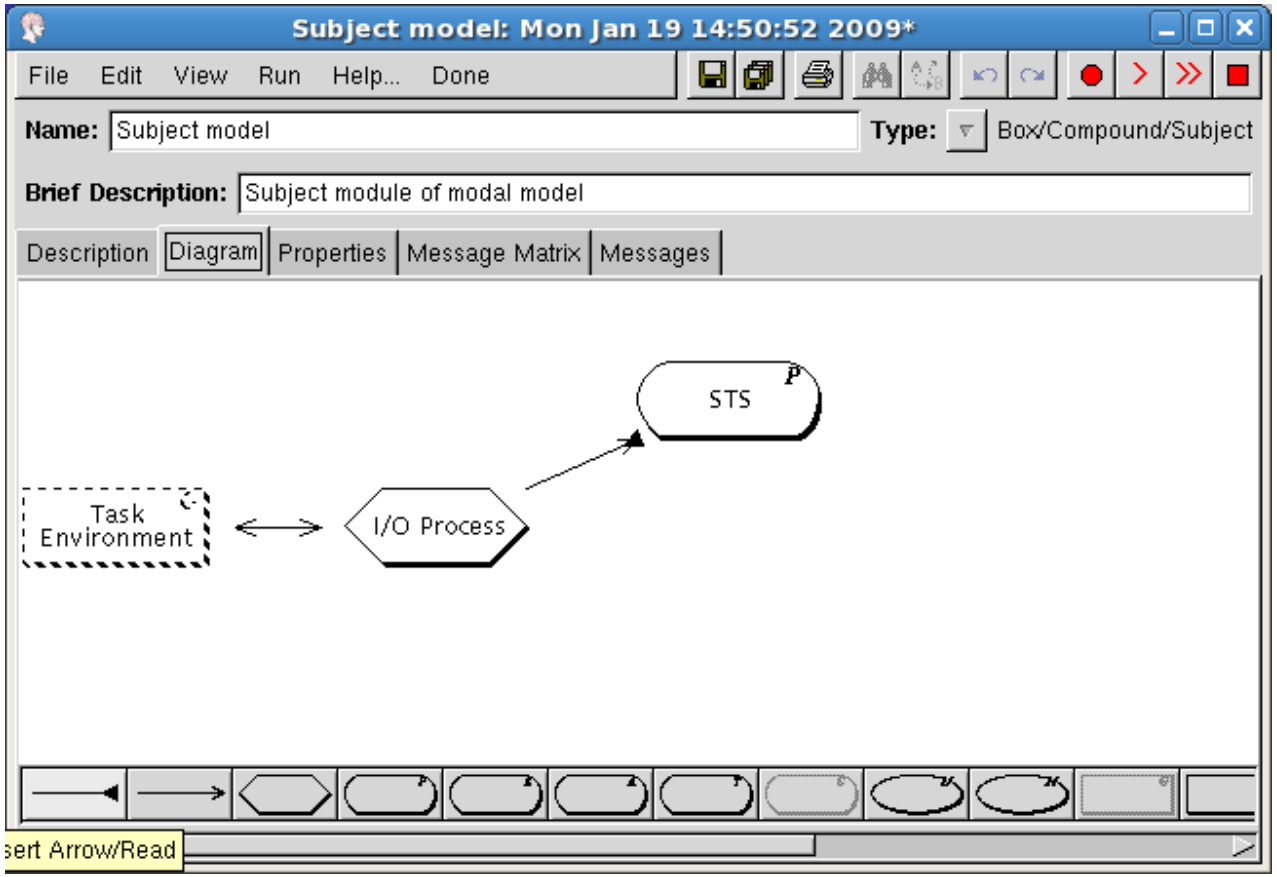
Building the Short Term Store

The rule to transfer words to **STS**:



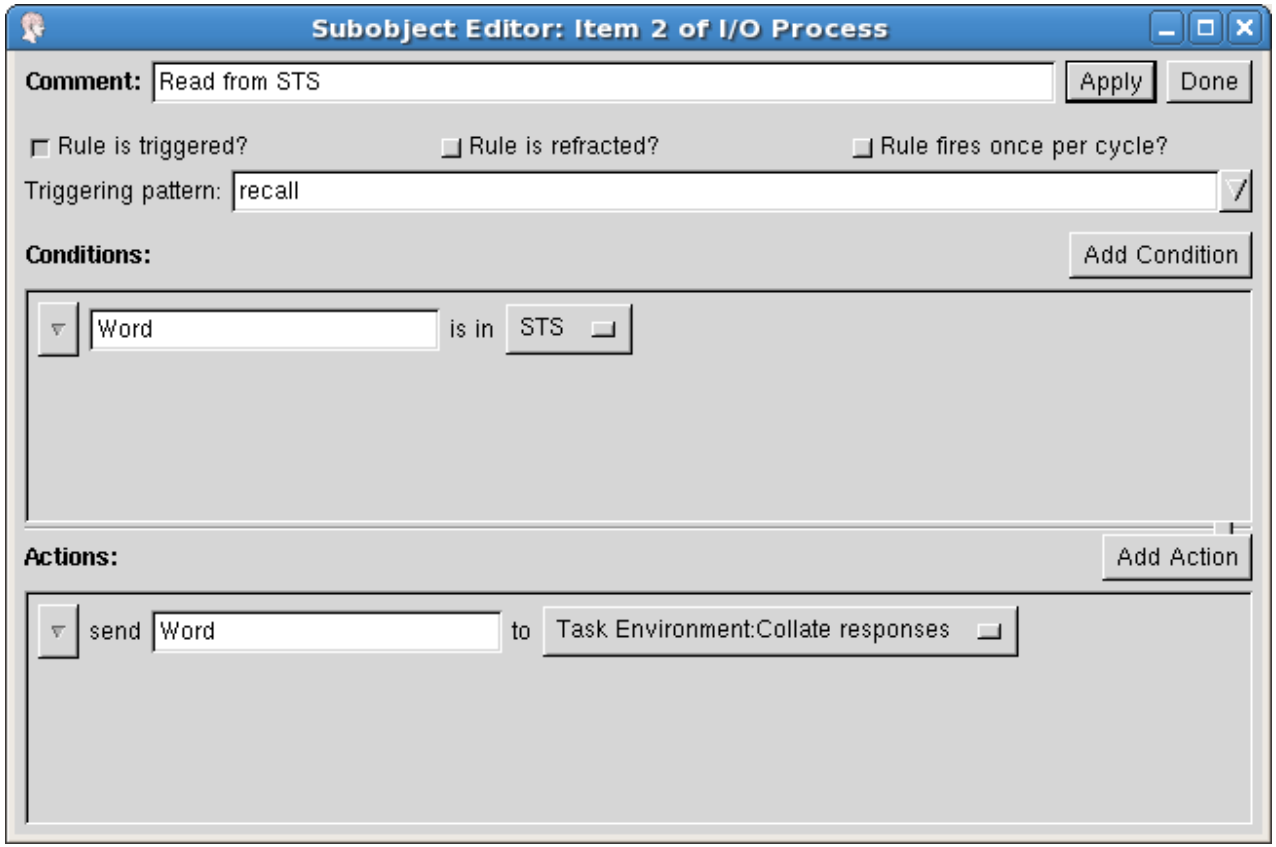
Building the Short Term Store

Add a read arrow from **I/O Process** to **STS**:



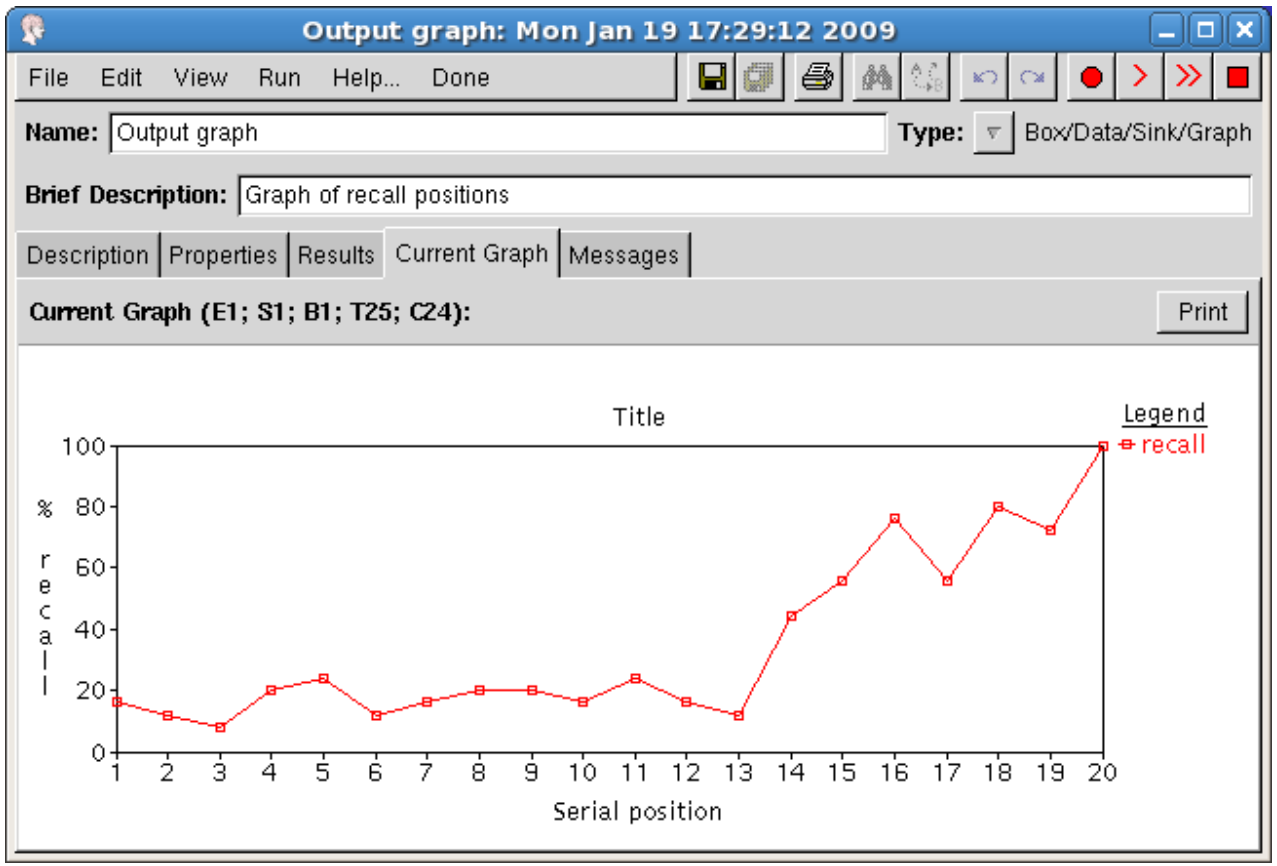
Building the Short Term Store

The rule to recall words from **STS** (use Add Condition → match):



Building the Short Term Store

Recall graph now shows a recency effect (output is for 20 trials):



Building the Short Term Store

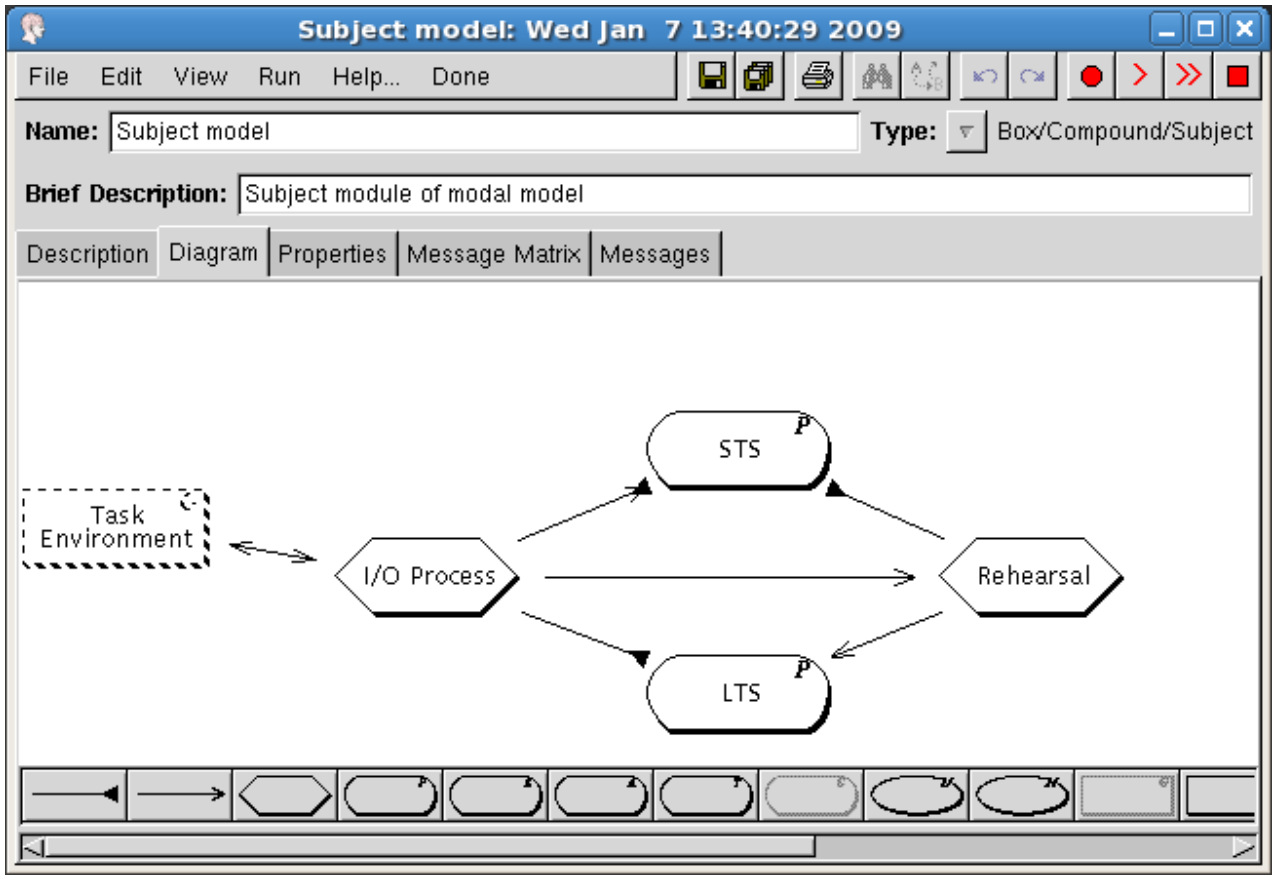
- What causes the recency effect? If we changed the properties of **STS**, could we change the shape of the graph?
- Watch the Messages view of **Input/Output**. What happens there now when you run (or single-step) through a trial?

Adding the Long Term Store

The modal model also includes:

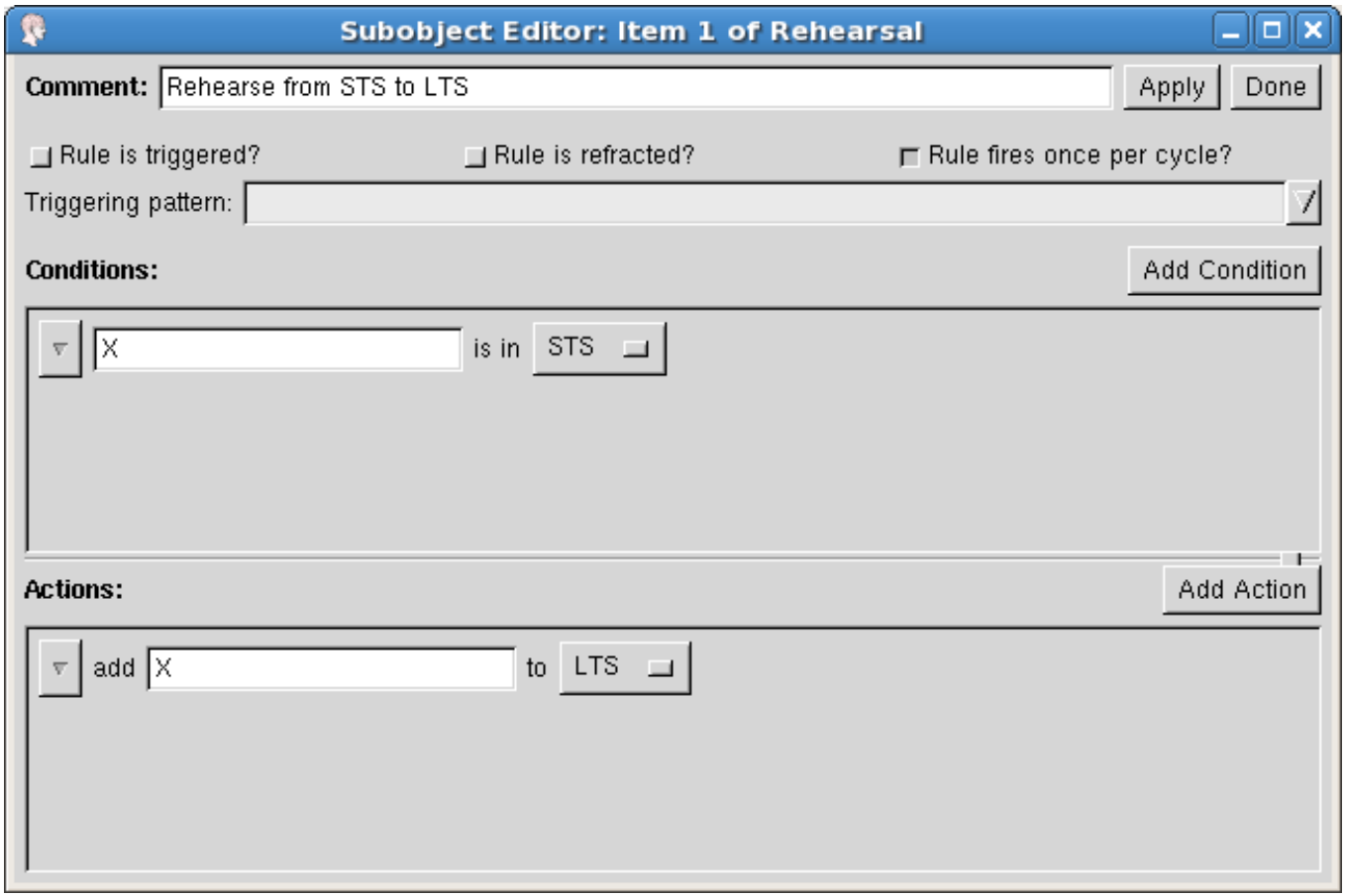
- a long term store (LTS);
- a rehearsal process to transfer information from STS to LTS;
- the possibility to recall from either STS or LTS.

Adding the Long Term Store



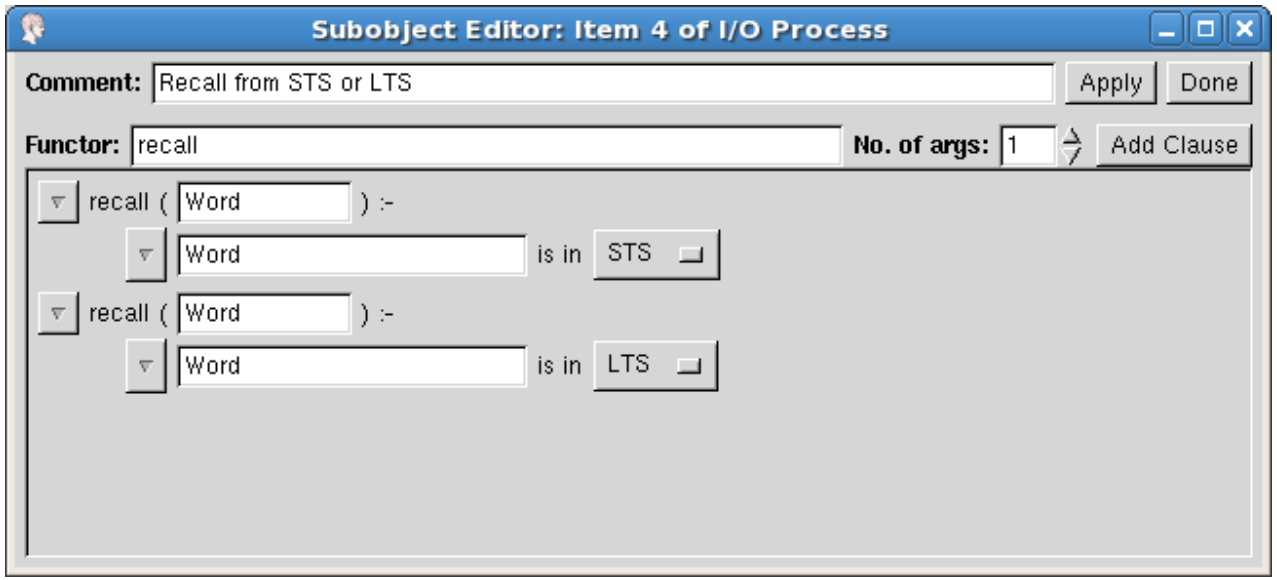
Adding the Long Term Store

The rehearsal rule:



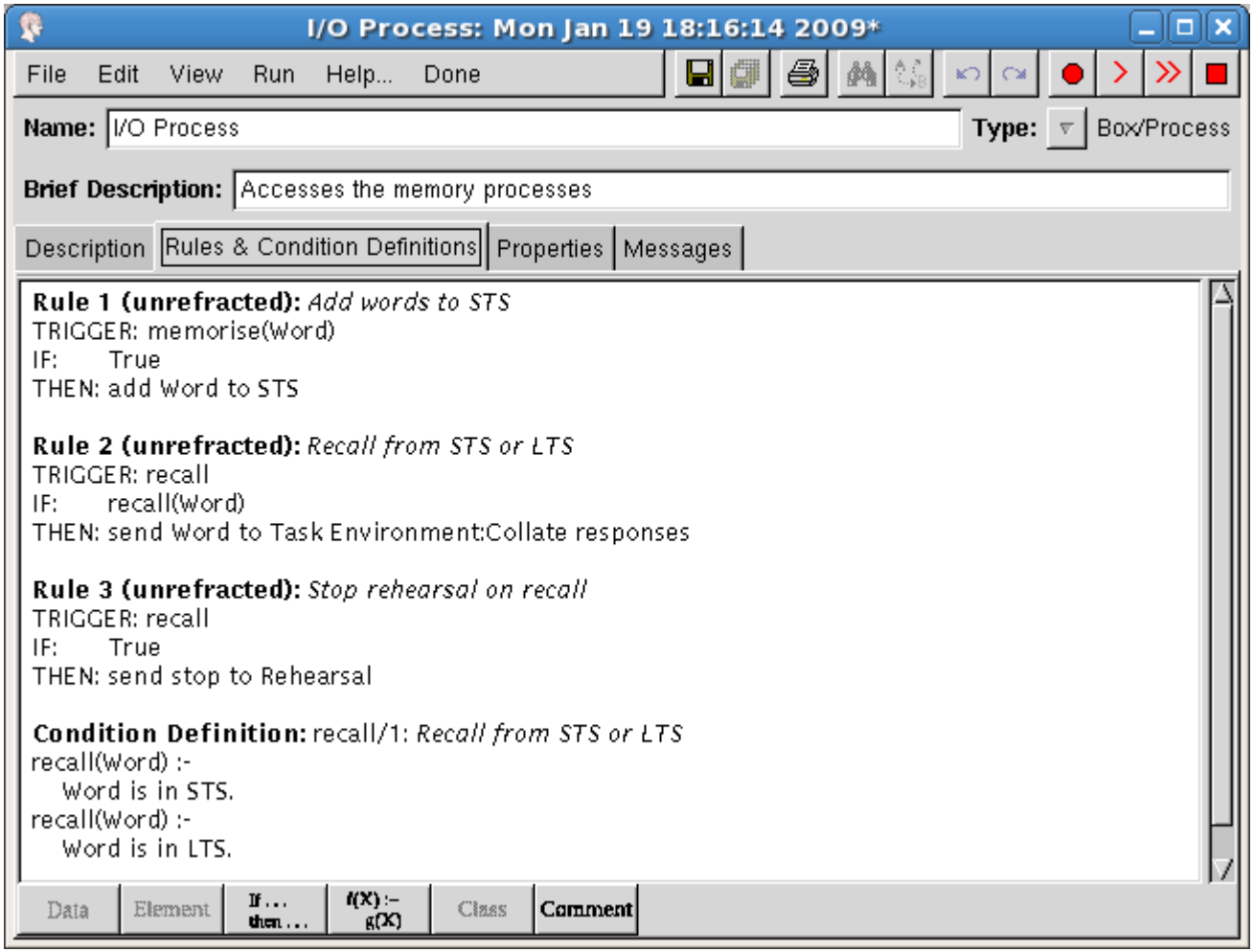
Adding the Long Term Store

To recall from either **STS** or **LTS**, we define a new condition by clicking the `f(X) :- g(X)` button and editing the definition:



- Read `:-` as 'if': `recall(Word)` is true if `Word` is in **STS**.

Adding the Long Term Store

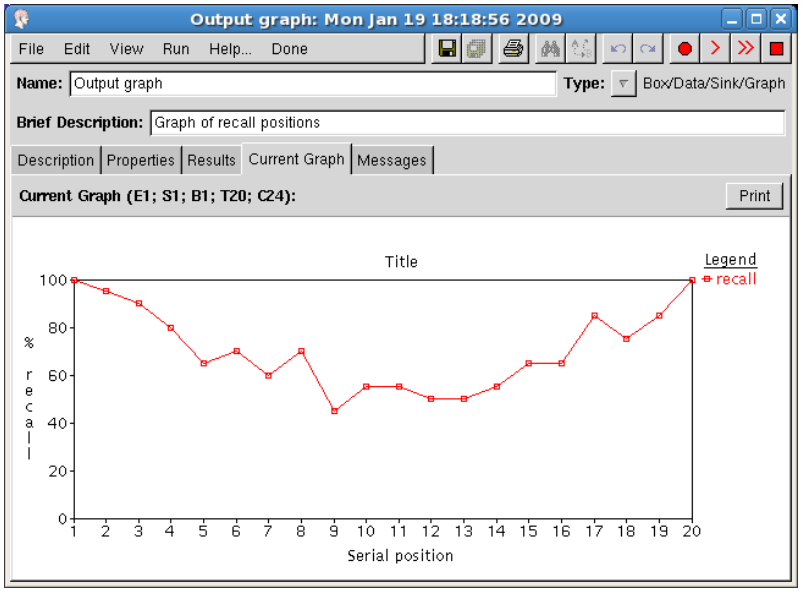


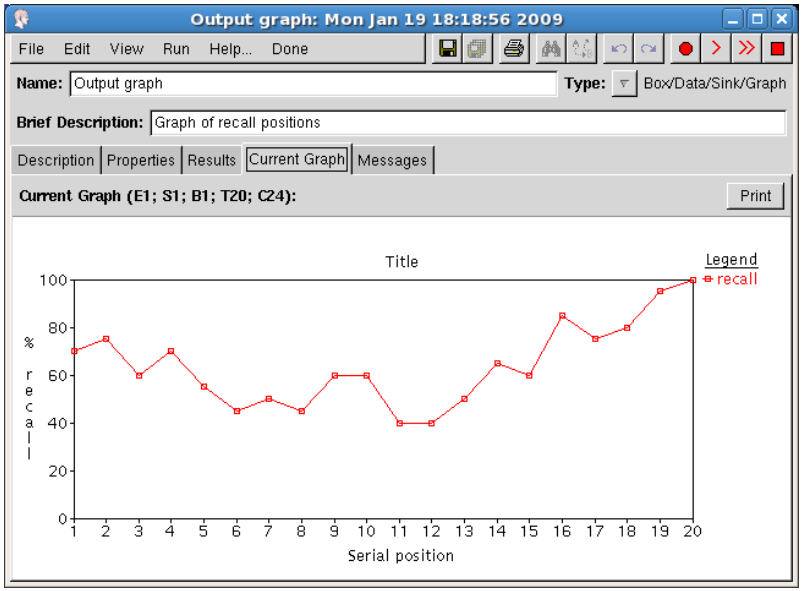
Adding the Long Term Store

- What causes the Primacy Effect?
- Monitor the Messages view of the **I/O Process**. Why does the model sometimes recall the same word twice in the same trial?

Adding the Long Term Store

The current output (left) still doesn't match the output from the intro (right). What is different? Why?



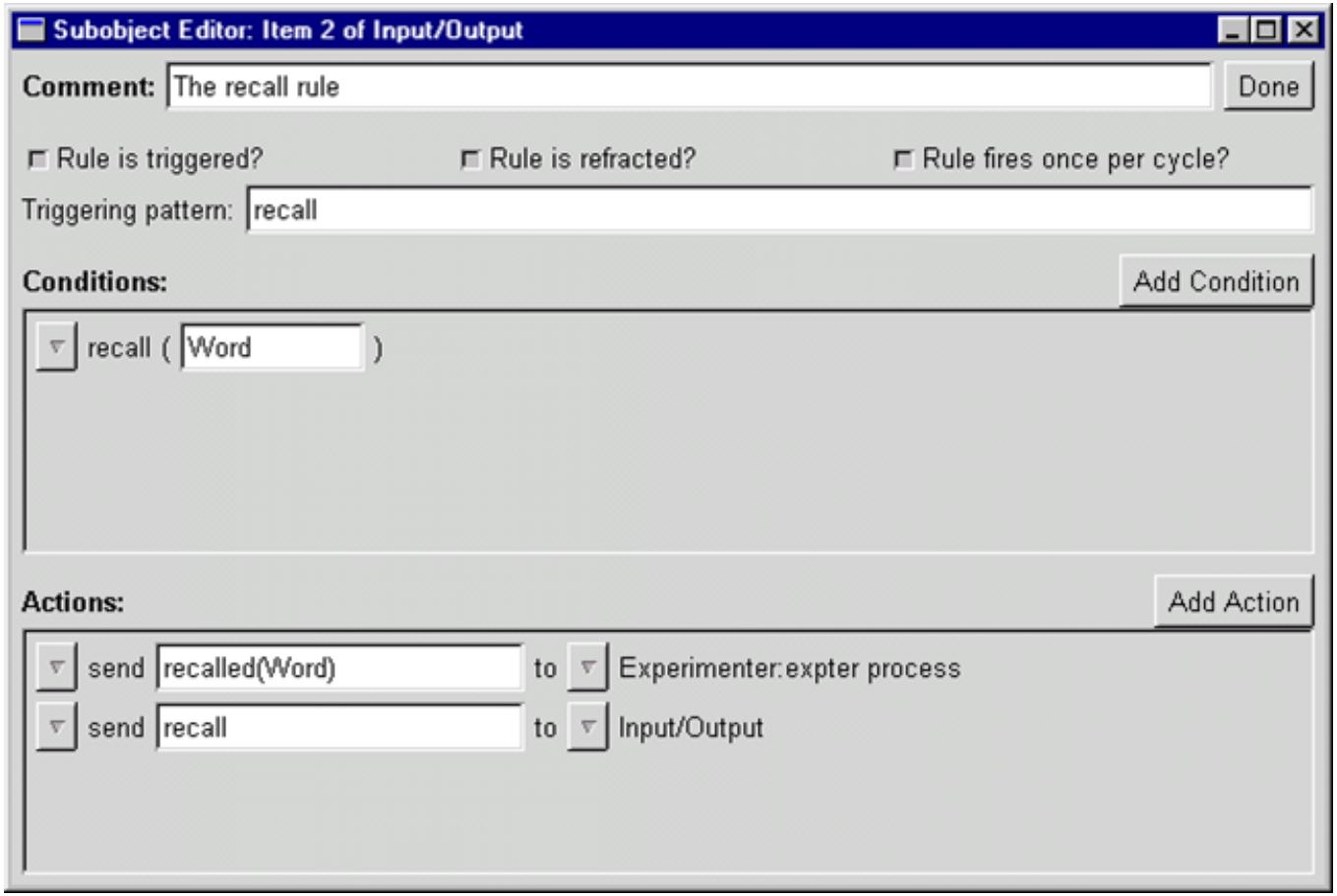


Decay, Time, and Rehearsal

- Add decay to **LTS**. Explore different decay rates.
- Change the rehearsal rate by adding another copy of the rehearsal rule.
- All memorized words are currently recalled in parallel. Make the recall process serial.

Decay, Time, and Rehearsal

The serial recall rule:



Decay, Time, and Rehearsal

- Explore the effect of the Access property of each buffer. Play with these (and other) parameters to see how they affect the model's behavior.
- The Experimenter system is written using standard Cogent. Try to discover how it works.
- Go on to develop the model into something substantial.

References

Cooper, Richard P. 2002. *Modelling High-Level Cognitive Processes*. Lawrence Erlbaum Associates, Mahwah, NJ.