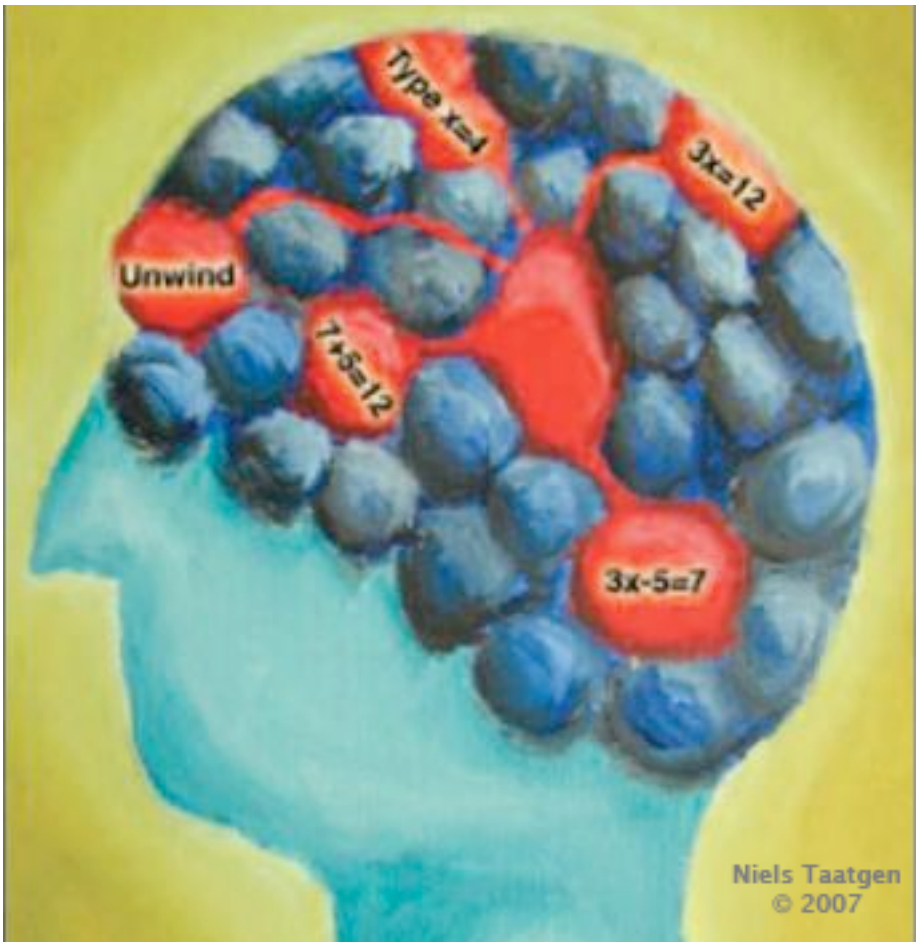

ACT-R:

The Java Simulation and Development Environment

Dario Salvucci
Drexel University

Introduction

The new ACT-R system provides an implementation of the ACT-R cognitive architecture (Anderson, 2007) in the Java programming language. The system can be used at three levels:

- **Beginners**. For users learning ACT-R or writing simple models, the system includes an easy-to-use application that wraps the core ACT-R implementation in an integrated editor and simulation environment. The user can write ACT-R models and immediately run them and inspect the resulting simulation traces. When used in this way, the system requires no Java programming whatsoever.
- **Intermediates**. For users wishing to code their own custom tasks, the distribution contains a Java archive file that can be included as an external library into a new code project. The user codes a task environment by extending the built-in classes for the task interface and associated components.
- **Advanced Users**. For users wishing to modify the ACT-R architecture itself and/or the standard development environment, the full source code is available under an open-source license.

The following sections detail each of these three uses.

The ACT-R Application

The distribution includes a standalone application for Macintosh OS X and a double-clickable JAR file for Windows and UNIX platforms. The Macintosh application has a standard OS X look and feel, and can be opened by double-clicking the application or one of the sample model files. On Windows and UNIX, the "ACT-R.jar" file can be opened by double-clicking, or can be opened from the command line by executing:

```
java –jar ACT-R.jar
```

in the file's directory.

On startup, the application opens a window with a new editor (left), task panel (upper right), and output panel (lower right). The user can immediately start entering ACT-R model text if desired. Alternatively, the user can open one of the tutorial files included with the system.

The model editor provides the standard set of features (cut/copy/paste, undo/redo, etc.) as well as automatic syntax highlighting and indentation. Settings for the font and font size, highlighting, and indentation can be modified in the Preferences dialog box.

The ACT-R model syntax generally follows the same syntax as the canonical LISP version, with a few differences. Because the task code is no longer in the model file, the model needs to specify the task being performed. This is done as follows:

```
(set-task "actr.tasks.tutorial.U1Count")
```

Many of the tutorial models include such statements to define the task. Models are not required to specify the task, however—for instance, if the model does include perceptual or motor elements and thus does not interact with a task interface.

Another difference is that arbitrarily LISP code is not possible within the model. Nevertheless, !eval! statements are allowed in two ways. The standard system can itself handle simple arithmetic "(= 2 =var)". For more complex functions, the system allows the user to provide code to evaluate custom statements; this will be discussed in the next section.

Once a model is loaded, it can be run in two ways. The standard "Run" command runs the model simulation in real time or some multiplicative factor of real time; the factor can be set using the arrow next to the Run button on the toolbar. The "Run Analysis" command runs the model some number of iterations, either as specified by the task or as specified by the user using the arrow next to the Run Analysis button. This latter command is intended for data fitting and generally outputs the results of analyzing the set of iteration runs.

For standard runs of a model, the system provides a preliminary set of ways to examine the model's current state, namely in printing the buffer state, the "why not" trace, and/or the contents of declarative memory. All these functions are available in the Output menu.

The application reads and runs two kinds of files. The first, a standard ACT-R model, has the extension ".actr". The second, a batch model file with the extension ".batch", lists the name of model files one per line; all model files must be in the same directory as the batch file. When a batch file is run, each model file is loaded and run, and an evaluation "score" (dictated by the task implementation) is given for each model (e.g., the correlation between its behavior and human behavior).

Developing Custom Tasks

While the standard application is useful for writing models of basic tasks, most intermediate and advanced users will require specification of their own custom tasks. Development of custom tasks requires programming, and we recommend the use of the Eclipse IDE (http://www.eclipse.org/).

To use Eclipse (or a similar IDE), the user should start by creating a new project and including the "ACT-R.jar" file as an external library. In Eclipse, after creating a new project, the user can right-click on the project name and edit the Properties of the project: under Java Build Path > Libraries, Eclipse allows a user to add the external JAR file "ACT-R.jar" as a new library.

Next, the user can create a package for the task code. We recommend including a prefix of "actr.tasks." for package names (e.g., "actr.tasks.mytask"). Finally, within this package, the user can create a class to define the custom task. For example, the new class might look as follows:

```
package actr.tasks.mytask;

import actr.task.*;

public class MyTask extends Task
{
    public MyTask ()
    {
        super();
        ...
    }
}
```

The custom task class should extend the base class actr.task.Task.

In the constructor, after calling super(), the code can create the task interface. The actr.task package contains several useful components for placing on the interface, most importantly TaskButton (for

push buttons) and TaskLabel (for simple text labels). Custom components can either extend of the existing components or can implement the TaskComponent interface. All components should then be added to the task interface. Note that all task components extend their close cousins in the Java Swing package, which includes a rich set of features for laying out interfaces, handling events, etc.

The example below shows a larger example of the MyTask class that includes a label and a button. The button specifies a method to handle mouse clicks (which in this case simply changes the text of the label).

```
package actr.tasks.mytask;

import actr.task.*;

public class MyTask extends Task
{
    public MyTask ()
    {
        super();

        final TaskLabel label = new TaskLabel ("Hello", 100, 100, 50, 20);
        add (label);

        TaskButton button = new TaskButton ("Press Me", 200, 200, 100, 50) {
            public void doClick() {
                label.setText ("Goodbye");
            }
        };
        add (button);
    }
}
```

In addition to the constructor, the actr.env.Task class contains a number of methods that can be overridden to further specify the task. These include:

- **start()**. Initializes the task interface and settings.
- **update()**. Updates the task interface given the current time.
- **addEvent(), addUpdate(), addPeriodicUpdate()**. Schedules events and updates for the task.
- **processDisplay()**. Updates the model's representation of the task display.
- **addAural()**. Adds an aural sound to the environment.
- **typeKey()**. Handles a task keystroke.
- **eval(), bind()**. Provides a custom implementation of LISP-like evaluations.
- **analyze()**. Analyzes a batch of tasks and, if desired, outputs the analysis.
- **analysisIterations()**. Gets the number of model iterations to perform during an analysis.

Please refer to the javadoc documentation that comes with the code distribution for more information.

Once the custom task class is defined, a model can use this task by specifying the task using the set-task command, including both the package and the class name in the full specification. For example, a model would refer to the sample task above as follows:

```
(set-task "actr.tasks.mytask.MyTask")
```

Developing the Architecture and Environment

To modify the underlying ACT-R architecture and/or the development environment, it is necessary to download and work with the full source code. The ACT-R source code is available for download on the project web site:

```
http://cog.cs.drexel.edu/act-r/
```

The source code is made available under the MIT open-source license. (The Java Look & Feel icons used in the application abide by their own license; see the license file in the src/resources/ directory.)

As for developing custom tasks, we recommend the use of the Eclipse IDE for source code development. Please contact the author if you have questions or issues regarding the source code.