Using Java to Provide Cognitive Models with a More Universal Way to Interact with Graphical User Interfaces

Farnaz Tehranchi[1]✉, Frank E. Ritter[1]

[1] Pennsylvania State University, University Park, PA, USA
{farnaz.tehranchi,frank.ritter}@psu.edu

Abstract. We present JSegMan, an approach to extend interaction in the ACT-R cognitive architecture. JSegMan allows cognitive models to interact directly with *any* application on a PC. JSegMan also simulates human visual attention movements. This work allows the behavior of models to be compared more directly to human behavior by using the same uninstrumented interface. This work also provides direct support for automated user interface testing and closed-loop system control. Furthermore, a new data structure—visual patterns—has been introduced to provide a more realistic representation of the world. We tested JSegMan by using it with an existing ACT-R model of a spreadsheet task, the Dismal model. Because the model interacted directly with a spreadsheet, we found defects in the Dismal model and resolved them. The revised model more accurately predicted a 20-min task while performing the task.

Keywords. Cognitive model; Cognitive architecture; Human-computer interface, interaction; Perception and motor output; Computer vision; Simulated eyes and hands.

1 Introduction

Cognitive architectures are infrastructures for cognitive science theory and provide computational frameworks to execute theories. They are languages specifically designed for modeling, such as Soar [1] and ACT-R [2]. We use ACT-R to develop cognitive models because it has a useful tutorial and reference manual, and is easily extendable through Lisp. We present an interactive approach that can move the cursor, click, and type in any application on a PC, and move the model's visual attention.

Cognitive models that interact with interfaces could be more useful in HCI [3-5] if they could intent with more tasks. To achieve this, cognitive models should be able to pass commands to the task environment and have access to the information on the screen to interact. With these abilities, cognitive models will be easier to develop and apply. This approach can be called a Cognitive Model Interface Management System (CMIMS), which is an extension of User Interface Management Systems (UIMS) [6].

We first provide a summary of suggestions from previous work about how models interact with the world. We will then describe the design of our JSegMan system that is independent of interface design and explain how it can be used to work with the

Dismal spreadsheet task and an existing ACT-R model [7]. The revised model with JSegMan predicted published response times more accurately. Finally, we provide conclusions, lessons, and insights.

2 Previous Systems

There are several approaches used by modelers to interact with the world. These include (a) modifying existing interfaces and creating interfaces in a special tool, (b) defining a communication protocol to bond a cognitive architecture process with another process that can implement actions, and (c) interacting with the operating system to generate commands and parse the screen bitmap. Previous systems illustrating these approaches provide lessons for our work.

The approach of modifying the interface or working in an interface language has been introduced in ACT-R/PM [8]. ACT-R/PM allows ACT-R models to interact with interfaces built within a special Common Lisp window. Cognitive Code is another approach that focuses on ACT-R devices—extensions of the ACT-R framework that provide external environments with which the models can interact. A successful version of this method is Salvucci's driver model [9]. He also introduced a new implementation of ACT-R in Java that can be used as a library in other ACT-R projects [10, 11]. Here, the ACT-R motor and vision module have been hard-coded, not configurable, and limited to a Java applet in the Cognitive Code.

For the second approach, Ong provides MONGSU as a general solution to the problem of connecting ACT-R to wide-ranging software programs [12]. His work proposes the idea of communicating between two processes through a UNIX socket. However, establishing the socket connection requires an additional application and advanced knowledge of programming. Hope et al. [13] presented a simplified interaction scenario between cognitive models and task environments through a JSON network interface. To use this connection, a new ACT-R device module (JNI module) for handling the communication must be added to the cognitive model. This method has been tested in different environments, such as the REACT-R module that uses the JNI module to connect ACT-R with a Unity 3D simulation [14]. Yet, the current JNI API version does not support all ACT-R functions from the motor and visual modules and does not have an independent interface. Thus, it is more useful for advanced users.

The third approach, SegMan, was an approach to interact with interfaces based on operating system calls [15-17]. It generated motor commands and injected them into the operating system event queue, and it parsed the screen bitmap to find objects. It was never fully completed but was used by several models, including ones that drove simulated cars [18], robots [15], and a variety of online systems including driving games, robot operation, and other screen-based interfaces [16, 17].

Our previous work [19] tried to provide a connection to any task environments in Emacs. We implemented pressing a key, moving a cursor, clicking a mouse, and moving attention. Our approach was independent of ACT-R model scripts. The eyes and hands method only worked on Emacs-based tasks. Emacs restrictions prevent this work from being a generic solution strategy.

Other approaches such as ACT-R/E [20] added new modules to ACT-R or modified ACT-R's modules. Neither ACT-R/E's technical details nor its re-usability is formally described.

In summary, the motivation for these simulated eyes and hands is to allow a model to interact with the same task that human subjects have to correspond more closely to human performance. Additionally, why do it separately for every project? Users could modify each application, but then a model of the eye and hand would have to be created in an awkward language and repeatedly. What we will do, however, is to create a way to interact with all interfaces using an extended Java library to input motor commands (keystrokes and mouse moves) and also a primitive vision system to see objects on the screen using pattern recognition and other computer algorithms.

3 The JSegMan Structures

The primary process of perceptual-motor for JSegMan is implemented in Java (Figure 1). We used two Java packages: (1) Robot and (2) Sikuli [21]. JSegMan uses Emacs as a coordination system, but it does not otherwise depend on Emacs functionality.

In the Shell environment, both ACT-R and Java processes can be run individually by passing arguments to each other (see Figure 1). When the Java process is running, the ACT-R process pauses and resumes when Java finishes the task. Therefore, the Java task interaction times will not affect the ACT-R response time. The Java process performs the visual and motor actions and gets feedback from the task environment. Although JSegMan's Java process verifies the correctness of the perceptual-motor function, JSegMan is still an open-looped control system because its ACT-R process does not wait for verification from the Java process nor check that it was successful.

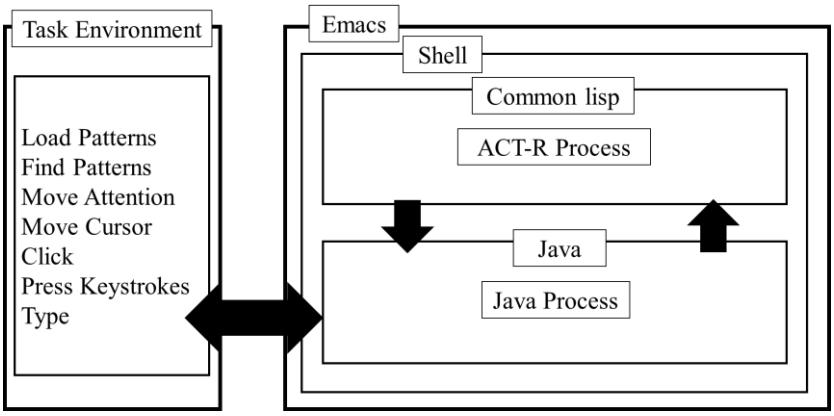


Fig. 1. Main components of the JSegMan approach/system.

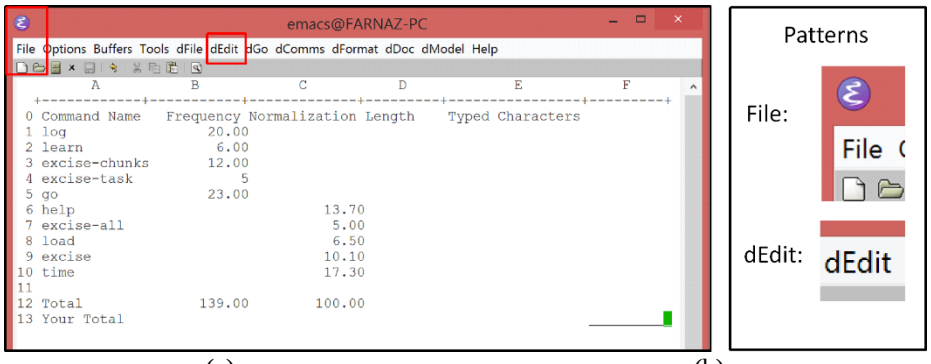
3.1 Simulated Hands

The hand's simulation produces keyboard and mouse motor abilities. The ACT-R Motor module interacts with the computer by pressing keys and moving and clicking the mouse. Creating these capabilities was difficult to do within Emacs

while using Emacs. While we still think it is possible, for generalizability and ease of use, we started to investigate other approaches to simulate. We found that Java has useful pre-implemented algorithms that can be used to simulate the interaction. The Java Robot package implements actions for the ACT-R motor module. Moving the mouse pointer as a surrogate for moving ACT-R's attention, clicking a mouse, and pressing keystrokes are handled by this package. Without using any user interface layer, we can directly get the mouse screen-x and screen-y locations. Figure 1 shows that the three motor module commands can be handled by the Java process (move cursor, click, and press keystrokes). The ACT-R process makes a call to the Java process with two input arguments. The first argument is a JSegMan function for the motor module; the second argument is the function parameter. For instance, the ACT-R process passes the *presskey* function and a letter (as a string) to the Java process to handle keypresses in the task environment.

3.2 Simulated Eyes

JSegMan uses patterns to recognize visual scenes and applies a pattern matching algorithm. Figure 2 demonstrates different patterns in the Dismal window, spreadsheet task environment in the Emacs. A pattern is a representation of an object in the visual scene. Patterns are a combination of visual location and visual object chunks. ACT-R can access all required objects of the current visual scene by defining them as independent patterns. Patterns should be significant enough for JSegMan to be identified uniquely. For instance, Figure 2b shows the differences between two patterns. Because *dEdit* is a unique menu icon and is only used in the Dismal window, its pattern can be small and only contains the dEdit word. However, the *File* pattern exists in all windows; to identify the *File* pattern just in Emacs window, the pattern should include more content such as the Emacs logo.



(a) (b)

Fig. 2. (a) The Dismal window in Emacs and (b) *File* and *dEdit* patterns.

Using patterns instead of existing ACT-R visual objects eliminates the dependency on features such as size and location for the cognitive model. Consequently, changes in the visual scene cannot affect patterns such as changes in screen resolution and size. The model can still find the target visual object because attention does not move by screen-x and screen-y. Instead, the model finds/re-finds the pattern in the visual

scene. JSegMan can simply get these features and update the visual representation in ACT-R from the Java process. This approach is independent of the task environment's system because the location of objects on the screen does not affect the recognition. Each pattern's existence will be checked by the Java process (*find pattern* in Figure 1). The current visual scene is a screenshot of a computer screen that is captured automatically by the Java process. ACT-R passes the *Value* slot in the vision object, the name of the pattern, to the Java process (*load pattern* in Figure 1). Finally, the task environment responds successfully to each of the requests made by the ACT-R model, and the ACT-R model is able to create and attend to objects within the dynamically changing visual scene.

The experimental environment is an application GUI that contains the information of visible objects such as labels, text fields, images, buttons, links, radio buttons, and toggle buttons. With Sikuli, we have access to all these objects and can define them as Java objects [21, 22]. With Sikuli, cognitive models can identify and control GUI components and also benefit from text recognition (OCR). More specifically, it uses screen bitmaps to search patterns to direct mouse and keyboard events in contrast to seeking screen locations, which may change during the experiment. Also, it utilizes a pattern-matching algorithm in OpenCV, an open-source computer vision library in which a pattern (small images) are compared against the overlapped image regions (the computer screen). For instance, to find the File pattern in Figure 2a, the pattern-matching algorithm slides through the screen and compares the overlapped areas with the File pattern. After comparison results are generated, the algorithm will select the best-matching item. Further details, installation document and instructions, and example models can be found on the project's website[1].

3.3 Applying JSegMan to the Dismal model

To test JSegMan, we worked with a spreadsheet task called the Dismal task and an ACT-R task model [7]. This task takes human subjects for about 20-30 minutes to perform. It is made up of 14 subtasks (e.g., click on the *File*) that are performed in a spreadsheet built in Emacs, but these tasks could quite literally now be done by the model using other unmodified spreadsheet applications such as Excel. In models like Dismal, the model's performance on the Dismal spreadsheet task is compared to the performance of participants. However, this comparison will not be completely realistic because the detailed trace of hand, fingers, and mouse movements has not been modeled in much detail.

In our analysis of the output of the model, the keystrokes and mouse moves, we found some missing actions. Including these missing actions increased the total task time. These modifications were independent of JSegMan. In this case, JSegMan helped fix system deficiencies. We were able to implement all the motor actions related to the keyboard in this model. By redefining some of the keystrokes, we made more realistic key press actions in ACT-R's virtual keyboard in ACT-R. Pressing a keystroke can happen when the model hand or finger is in the right position. For some keys that are not reachable by either the left or right hands, there must be a request to

[1] https://sites.psu.edu/ftehranchi/projects/

the motor module to adjust the hand position that was not included in the initial model. Besides proving the functionality of the JSegMan Dismal model, we were able to recognize the hand or finger re-position requirements that were not in the original model. The absence of these hand's movements was determined by JSegMan. We added these missing movements to the existing model [7].

Table 1 shows how the response time was affected by our modification while the model learns over four trials. As a result, with the JSegMan hands correction, the Dismal model better fits human data, has an increased correlation, and decreases mean square error.

Table 1. The mean task completion time (seconds) for four learning sessions for the Dismal task (N=30) [7].

Day	Human		Original Model		With JSegMan Hands Correction	
	M	SE	M	SE	M	SE
1	1366	60.8	1326	12.078	1338	12.06
2	894	26.6	891	6.175	893	5.144
3	727	25.5	693	4.496	700	6.207
4	659	22.7	594	5.775	603	4.35
Correlation			0.997		0.9978	
MSE			1747.5		1162.5	

4 Conclusion and Future Work

We present a solution to the challenge of communication between external task environments and cognitive architectures that previously required redefining interfaces for cognitive models. JSegMan provides simulated eyes and hands. It currently requires the original ACT-R model, the Robot and Sikuli packages in Java, and Emacs as glue. We augmented the manual and the vision module with operating level accessibility. The JSegMan approach will increase the usability, applicability, and accessibility of cognitive architectures. This article focused on models written with the ACT-R cognitive architecture, but this approach and system could be used by other architectures.

Further work remains. We need to implement a closed system control to check if the action takes place correctly in JSegMan and the eyes move with the hands successfully. This result illustrates that JSegMan also offers the ability to understand a model more accurately and adds to the capacity to understand the model. JSegMan needs to create multi-part patterns to interact with more complicated environments. In the future, we plan on offering an installation method that includes bundled versions of all dependencies, allowing near plug and play support with ACT-R. JSegMan components need to be expanded so JSegMan can observe the users, collect more realistic inputs, and thus better predict more complex, interactive human performance. Therefore, JSegMan can start to be a substitute for humans in the software testing process and can more seriously be considered as a software testing tool.

5 Acknowledgments

This work was funded partially by ONR (N00014-15-1-2275). David Reitter has provided useful comments on Emacs and Aquamacs (the Emacs version for Mac). We wish to thank Jong Kim who provided the idea for ESegMan, and Dan Bothell for his assistance wrangling ACT-R.

References

1. Newell, A.: Unified Theories of Cognition. Harvard University Press, Cambridge, MA (1990)
2. Anderson, J.R.: How can the human mind exist in the physical universe? Oxford University Press, New York, NY (2007)
3. Byrne, M.D., Kirlik, A.: Using computational cognitive modeling to diagnose possible sources of aviation error. International Journal of Aviation Psychology 15, 135-155 (2005)
4. Pew, R.W., Mavor, A.S. (eds.): Human-system integration in the system development process: A new look. National Academy Press., Washington, DC (2007)
5. Kieras, D.E.: Model-based evaluation. In: Jacko, J., Sears, A. (eds.) Handbook for human-computer interaction, pp. 1139-1151. Erlbaum, Mahwah, NJ (2003)
6. Ritter, F.E., Baxter, G.D., Jones, G., Young, R.M.: User interface evaluation: How cognitive models can help. In: Carroll, J. (ed.) Human-Computer Interaction in the New Millenium, pp. 125-147. Addison-Wesley, Reading, MA (2001)
7. Paik, J., Kim, J.W., Ritter, F.E., Reitter, D.: Predicting user performance and learning in human-computer interaction with the Herbal compiler. ACM Transactions on Computer-Human Interaction 22, 25 (2015)
8. Byrne, M.D., Anderson, J.R.: Perception and action. In: Anderson, J.R., Lebiere, C. (eds.) The atomic components of thought. Erlbaum, Mahwah, NJ (1998)
9. Salvucci, D.D.: Modeling driver behavior in a cognitive architecture. Human Factors 48, 362-380 (2006)
10. Salvucci, D.D.: Rapid prototyping and evaluation of in-vehicle interfaces. ACM Transactions on Computer-Human Interaction 16, 33 (2009)
11. Salvucci, D.D.: Integration and reuse in cognitive skill acquisition. Cognitive Science 37, 829-860 (2013)
12. Ong, R.: Mechanisms for routinely tying cognitive models to interactive simulations. U. of Nottingham. Available as ESRC Centre for Research in Development, Instruction and Training Technical report #21 (1994)
13. Hope, R.M., Schoelles, M.J., Gray, W.D.: Simplifying the interaction between cognitive models and task environments with the JSON Network Interface. Behavior Research Methods 46, 1007-1012 (2014)
14. Salt, L., Wise, J., Sennersten, C., Lindley, C.A.: REACT-R and Unity Integration. In: Proceedings on the International Conference on Artificial Intelligence (ICAI), pp. 31. (2016)
15. Ritter, F.E., Kukreja, U., St. Amant, R.: Including a model of visual processing with a cognitive architecture to model a simple teleoperation task. Journal of Cognitive Engineering and Decision Making 1, 121-147 (2007)

16. St. Amant, R., Riedel, M.O., Ritter, F.E., Reifers, A.: Image processing in cognitive models with SegMan. In: Proceedings of HCI International '05. Erlbaum, (2005)
17. St. Amant, R., Riedl, M.O.: A perception/action substrate for cognitive modeling in HCI. International Journal of Human-Computer Studies 55, 15-39 (2001)
18. Ritter, F.E., Van Rooy, D., St. Amant, R., Simpson, K.: Providing user models direct access to interfaces: An exploratory study of a simple interface with implications for HRI and HCI. IEEE Transactions on System, Man, and Cybernetics, Part A: Systems and Humans 36, 592-601 (2006)
19. Tehranchi, F., Ritter, F.E.: An eyes and hands model for cognitive architectures to interact with user interfaces. In: MAICS, The 28th Modern Artificial Intelligence and Cognitive Science Conference, pp. 15-20. (2017)
20. Trafton, G., Hiatt, L., Harrison, A.M., Tamborello, F., Khemlani, S., Schultz, A.: ACT-R/E: An embodied cognitive architecture for human-robot interaction. Journal of Human-Robot Interaction 2, 30-55 (2013)
21. Yeh, T., Chang, T.-H., Miller, R.C.: Sikuli: Using GUI screenshots for search and automation. In: Proceedings of the 22nd Annual ACM symposium on User interface software and technology, pp. 183-192. ACM, (2009)
22. Kasper, M., Correll, N., Yeh, T.: Abstracting perception and manipulation in end-user robot programming using Sikuli. In: Technologies for Practical Robot Applications (TePRA), 2014 IEEE International Conference on, pp. 1-6. IEEE, (2014)