CogSysIII Lecture 11: Enduser Programming

Human Computer Interaction

Ute Schmid

Applied Computer Science, University of Bamberg

last change July 10, 2007

Enduser Programming

- programming that is done by a software user who is not a programmer, presumably without any specific training in programming
- Goals
  - Make the power of computers fully accessible to all users
  - no limitation to the capabilities of the software they are provided
  - enable users without programming expertise to construct software

Methods

- Macro and scripting languages
- Recording and generalization
- Creating UIs to programming which make programming easier
  - visual programming: dataflow diagrams, flowcharts, or screen-layout systems
  - natural-language syntax: making the programming language look more like a spoken human language (e.g. Cobol and HyperTalk tried this strategy)
  - forms-based programming; using forms and dialogs (such as wizards) to create a program
  - programming by example: user provides examples of how a program should operate, either by demonstrating the steps or by showing examples of the inputs and outputs, and the system infers a program that would achieve those examples.
  - programming by demonstration: letting users perform their tasks and inferring programs that correspond to them

Typical Areas of Application

- Web Agents
- Geographic Information Systems
- CAD
- Text Editing Macros
- Macro generation in general
- For programmers: Relieve from routine programming, debugging
- Education, games
- see: Allen Cypher (Ed.), Watch What I Do – Programming by Demonstration, Edited by Allen Cypher, MIT Press, 1993
  Henry Lieberman (Ed.), Your Wish is My Command: Programming by Example, Morgan Kaufmann, 2001

Core Problem of PbD/PbE

- infer what loops and conditionals might be involved!
- may allow the user to select among several candidate programs that attempt to generalize the users demonstrated task into a program
- works well for very simple tasks
- may even work quite well in specialized domains where meaningful inferences can easily be made
- BUT: programming by demonstration certainly will not be replacing traditional programming for large-scale programs
- Relation to research in knowledge-based software engineering

Example 1: Eager

- by Allen Cypher (1993)
- Programming repetitive tasks
  - Inserting meeting times in a calender
  - Reformatting literature from quotes to italics
  - Making a list of the subjects of a stack of messages

Eager Example

- user selects and copies the subject of the first message, goes to the list, types "1. " and pastes in the subject
- navigates to the second message, selects and copies its subject, goes to the list, types "2. ", and pastes in the second subject
- After user navigates to the third message, the Eager icon appears, indicating that Eager has detected a pattern in the user's actions. Eager does not immediately begin performing actions; rather, it uses green highlighting to "anticipate" what the user is going to do next.
- Eager is anticipating that the user will select the subject of the third message
- User selects and copies the third subject, goes to the list, and Eager anticipates that the user will type "3. "
- After the user does this, Eager anticipates that the user will paste
- User does the paste and navigates to the fourth message, which is anticipated
- Now user is confident that Eager can perform the task correctly and therefore clicks on the Eager icon
- A dialog box appears, and the user selects the option "Finish The Task"
- Eager writes and executes a program which completes the task for the user, and automatically terminates

Eager Principle

- Minimal intrusion
- never asks the user for information; recording the user's ongoing actions
- user does not signal the start of an example, and never explicitly confirms or rejects a hypothesis; performing an action that matches the anticipation is implicitly confirming; performing any other action is implicitly rejecting.

Eager Method

- Representation of actions by highlighting objects, menu items, and text selections (Macintosh has a direct-manipulation interface)
  NO displaying of textual descriptions (Copy the button located at 20,120)
- Domain Knowledge about the structure of applications
  - Dealing with high-level information: e.g., not mouse click occurred at location (28,142), but click on button named "Phone"
  - E.g., knowledge about HyperCard objects which includes the fact that there is a last card in a stack, so Eager can infer that a loop that copies Card 1, Card 2, ..., should terminate when the last card in that stack is copied.
  - E.g., number called "button-location-X" is of type screen coordinate, so Eager uses a test for similarity that recognizes linear sequences of numbers within a tolerance of 4 pixels, to allow for inexactness in where users position objects on the screen.

Eager Method: Pattern-Matching

- commands are of the same type (e.g. they are both Copy Text commands),
- and objects on which the commands are performed fit into some regular pattern (e.g. Tuesday and Wednesday ; button #6 and button #7 ; the last word on line 4 of field "Address" of card 2 of stack "Rolodex" and the last word on line 4 of field "Address" of card 3 of stack "Rolodex").
- for integers: constants, consecutive integers, linear sequences, and linear sequences within a given tolerance (e.g. 50, 72, 91, ... is recognized as the sequence 20*i + 30, with a tolerance +/-2).
- for textual data: parsing of text into substrings of contiguous alphabetic characters, numbers, delimiters, and spaces; searches for constants, alphabetic or numeric order, changes in capitalization and spacing, and known sequences such as days of the week and roman numerals

Eager Method: Loop-Identification

- When two similar events (see pattern-matching) are found, Eager assumes that the second event marks the beginning of the second iteration in the loop.
- All of the events before it, back to the first similar event, are presumed to constitute the first iteration in the loop.
- Eager now monitors each new incoming event to see if it is "similar" to the corresponding event from the first iteration.
- If patterns can be found for each pair of events, Eager concludes that it has detected an iterative loop, and the Eager icon pops up on the screen.
- Based on the generalizations formed from these two iterations, Eager instantiates the next steps in the pattern and directs the application to highlight the appropriate items in green.

Eager User Study

- seven subjects were given three repetitive tasks to perform. The subjects were not given any information about Eager – they were simply asked to perform the three tasks.
- Recording of verbal protocolls
- users were generally able to understand what Eager was doing and to figure out how to use it without instruction
- Eager was able to detect the patterns in their actions, even though different subjects chose different strategies for performing the tasks, and some subjects performed the task somewhat differently on each iteration. For instance, one subject originally used the "Next" button to go to the end of the stack, and on a later iteration switched to using the "Last" command. Also, some subjects made and corrected minor mistakes, such as navigating past the desired card and then backing up.

Eager User Study: UI

- all subjects were uncomfortable with giving up control when Eager took over.
  - Eager now saves a copy of any stacks it is going to modify so that the user can recover from an incorrect automation,
  - a stepping mode was added, where Eager pauses for user confirmation before each action.
- Some subjects did not realize that the character that popped up on the screen was related to the items being highlighted in green. To remedy this, the icon now appears in green as well.

Eager User Study: UI cont.

- original icon showed a man sitting in a chair. When he anticipated an iteration correctly, he would begin clapping. Some subjects expressed confusion about the character, since they could not see what the representation had to do with automating repetitive tasks. The icon was therefore changed to the less evocative image of a cat – when it takes over, the cat is shown with its paw moving the mouse.
- Some subjects did not notice the icon when it first appeared, particularly if it appeared on a rich background. The cat icon now animates when it first appears.

Eager Limitations

- A limitation in using anticipation to communicate an abstract program to the user is that anticipation is unable to communicate the termination conditions for the program.
- A user expecting a PBD system to delete all of the documents in the current folder would be shocked to observe it blithely deleting all of the documents on the disk!
- At present, Eager is conservative and terminates when it reaches the end of the first structure in the PartOf hierarchy. It would be reasonable to query the user about continuing the iteration to the next level in the hierarchy.

Eager Limitations

- Cannnot handle conditionals, nested loops, repetitions that are spread out over time (e.g. inserting the date whenever you start a new letter), or tasks that include some steps that must be left up to the user.
- These restrictions help to constrain the sorts of patterns that the program must detect.
- If Eager's search fails to detect a match between an incoming event and the corresponding event in the previous iteration, it assumes that there is no repetitive task.
- In fact, the correct interpretation could be that there is a conditional branch in the program, or that there is an obscure pattern that must be left up to the user (e.g. deleting the names of people who are no longer friends).
- Allowing for these possibilities would greatly increase the ambiguity in generalizing and the complexity of pattern-matching.

Inductive Program Synthesis

- A more powerful technique to generalization than Eager
- Applied for learning XSL transformations with recursive template applications
- see KogSysII