

The Challenges of Implicit Programming by Example

Jean-David Ruvini
Bouygues e-lab
1 avenue Eugène Freyssinet
78061 St Quentin en Yvelines
jdruvini@bouygues.com

Categories and Subject Descriptors

H.5.2 [Information interfaces and presentation]: User Interfaces.

General Terms

Algorithms, Experimentation, Human Factors.

Keywords

End-user programming, programming by example, adaptive interfaces, machine learning.

1. INTRODUCTION

It is evidence that users of interactive applications perform a lot of repetitive actions and commands while manipulating their software. To offer end-users the possibility to automate these tedious repetitions, early interactive applications proposed script languages, allowing to write programs that can be later on invoked to perform a repetition within a single mouse-click or key-stroke. However, script languages are not widely used because they require efforts and programming knowledge that many users do not have and because writing (and debugging!) a program often takes longer than performing the repetition manually. These obstacles are often referred as the “just-in-time programming” problem [12].

A significant step toward alleviating repetitions has been achieved with Programming by Example [2], [10] (PbE, also called Programming by Demonstration). Built around the macro recorder metaphor, PbE systems let the user demonstrate through examples what the task to automate should do and create a program (containing variables, iterative loops or conditional branches) from observing this demonstration. Although spectacular progresses have been made in PbE (see for example SMARTedit [8]), the old macro recorder metaphor where the user is explicitly involved in the program creation process (he has to start and stop the recorder) is still dominant.

In this paper we focus on what we call implicit Programming by Example which aims at offering end-users the possibility to automate complex repetitions with the minimum of user’s intervention and effort. More precisely, implicit PbE is based on two key ideas. First, it avoids the macro recorder metaphor as the

user does not have to start and stop a recorder; the repetitions are automatically detected from observing the user’s behavior and programs to automate them are automatically inferred. Second, it makes use of the assistant metaphor as the system anticipates repetitions and automatically offers assistance when appropriate. Implicit PbE solves the just-in-time programming problem as the user never enters a programming process and is automatically suggested help when appropriate.

Keeping user’s effort as low as possible is important for several reasons. First, there are some situations where the recorder approach is just inapplicable. This is the case for instance in the pervasive computing model where the user might not be fully aware of the computers surrounding him. Second, the user might not be aware of all the repetitions he is performing. This is the case with repetitions that spread over several work sessions or several days. Third, inferring the correct program from a few examples is difficult. As a consequence, explicit PbE system can rarely guarantee to the user if and how he will be paid back for his efforts, increasing the risk of frustration. Alternatively, since users do not make any effort in implicit PbE, every correct suggestion the system makes is bonus. Furthermore, experiments [14] have suggested that, as far as the system globally achieves reasonable performances, incorrect suggestions do not hurt.

However, we do not claim that explicit PbE is doomed to failure since many applications require the “teacher” metaphor. Also, an advantage of explicit PbE is that it can provide automation as soon as the second repetition whereas implicit PbE requires at least two iterations of a repetition before offering any automation.

The goal of this paper is to propose a state-of-the-art in implicit PbE. It is structured as follows: First, we briefly review related work in the area of PbE. Second, we identify the technical difficulties of implicit PbE, describe the solutions proposed so far and suggest directions for future research. In the third part, we review some metrics to measure the effectiveness of implicit PbE systems. The paper concludes with a summary.

2. IMPLICIT VS. EXPLICIT PBE

Programming by Example found a broad audience application in interactive software through macro recorders. Although macro recorders are still present in most of today’s applications, they are not widely used. Potter [12] identified a set of obstacles, collectively referred as the just-in-time programming problem, that typically dissuade users:

1. **The effort of demonstrating the macro** to record, including judging if recording a new macro is appropriate and the efforts the user might expend correcting inevitable errors.
2. **Limited computational generality** since macro recorders perform rote learning, making the macro useless in situations not exactly identical to the situation in which it was recorded.

3. **The effort of invoking the macro** including judging if it is appropriate and applicable, remembering how to invoke it.
4. **Risk** since demonstrating the macro may take longer than expected or longer than performing the repetition manually.

Solutions to solve these problems came from two correlated areas of research, namely PbE and Predictive Interfaces. PbE research has mainly focused on solving the computational generality problem while Predictive Interfaces research focused on the demonstration, invocation and risk problems. However, the distinction between PbE systems and Predictive Interfaces has blurred and we claim that the distinction should now be made between explicit PbE systems which require explicit and active participation from the user and implicit PbE systems which minimize as little as possible user's intervention and efforts. We propose below a brief review of two decades of research in PbE and Predictive interfaces based on this distinction. For a broader view, the reader can refer to [2] and [10].

Explicit PbE systems are designed around the "teacher" metaphor where the user teaches to the computer through examples how to perform a specific task and are able to learn sophisticated programs containing variables, conditionals, recursions or iterations. Recent examples are Toontalk [4] a programming environment for children implemented as a video game, and Grammex [9], an agent that learns recursive grammar rules for text parsing. Explicit PbE also includes systems that solve some of the other just-in-time programming problems. For instance SMARTedit [8], designed to automate repetitive text-editing procedures, lets the user perform manually a few iterations of a task, learns from these iterations and is able to offer to complete the task automatically. It is an explicit PbE system because it requires the user to start and stop a recorder but it also minimizes invocation efforts since it automatically offers some automation when appropriate.

Implicit PbE systems minimize user's efforts and intervention. Building on Predictive Interfaces research, they typically draw on machine learning to learn correlations between situations the user encounters and the corresponding actions he performs, and assist him by suggesting to perform automatically some part of user's work. There is no special learning or recording mode the user has to activate and he implicitly trains the system by simply using it. They avoid any demonstration effort and minimize invocation efforts using a suggestion mechanism. Famous examples are, among others, Maes's assistants [11] which advise users for some application specific operations like managing mails or selecting articles in news or WebWatcher [1], an assistant for the world wide web, that suggests links of interest to the user. However, these systems offer automation for simple tasks only. Modern implicit PbE systems have contributed to increase computational generality of implicit PbE systems while keeping user's effort to a minimum. For instance Eager [2], an assistant for the HyperCard environment, is able to detect the first iterations of a loop in the user's behavior and to propose to complete the loop until a "condition" is satisfied. APE [13], an assistant for a programming environment, offers to automate repetitive sequences of actions or commands, to complete loops or to write repetitive portions of code. It extends Eager in that it offers automation even if the different iterations of these repetitions are not consecutive in the user's history of actions.

3. TECHNICAL SPECIFICITIES

Implicit PbE systems watch over the shoulder of the user, detect repetitions in his behavior and automatically offer to automate these repetitions when appropriate. The task of an implicit PbE system can be split in two sub-tasks: first, learning what to automate (the repetitions) from observing the user's behavior; second, learning when to make a suggestion to the user. In the next sections we detail these two tasks and describe the approaches proposed so far.

In this paper we call "history of user's actions" the sequence of the actions the user performs in a work session. Actions can be low level events like key-strokes and mouse-clicks or high level events corresponding to specific domain dependant operations (like opening a file). This flat model of the user's behavior is common-place in PbE.

3.1 LEARNING WHAT TO AUTOMATE

The advantage of the macro recorder approach is that when he presses the start and stop buttons, the user explicitly marks the beginning and end of one (or more) iteration of the repetition and thus provides a useful pattern for predicting further iterations. The main difficulty of implicit PbE is that the system has to identify the iterations without any explicit intervention from the user, particularly when these iterations are similar but not exactly the same. We identify several kinds of repetitions of interest in implicit PbE:

- **Constant:** all the iterations of the repetition are identical. Repeatedly inserting a table with a specific number of rows and columns in a text processor is an example:
 - On the Table menu, point to Insert table.
 - In the Number of Columns list, click 4.
 - In the Number of Rows list, click 10.
 - Click OK.
- **Loop:** all the iterations comprise the same actions but are repeated over a set of objects. Formatting all the pictures in a document is an example:
 - For each picture P in current document
 - Select P.
 - Click Add border.
 - Click Center.
- **Nested loops:** iterations are identical but embedded in two nested loops. Example:
 - For each document D in current folder
 - Edit D
 - For each picture P in D
 - Select P.
 - Click Add border.
- **Conditional loop:** iterations are identical but are repeated over a set of objects exhibiting a certain property.
 - For each message m in MailBox
 - Select m.
 - If SenderOf(m) is Tessa
 - Then Forward m to Dan.
- **Variable loop:** the iterations are repeated over a set of objects and depend on a property of the object.
 - For each message m in MailBox
 - Select m.
 - Save m to 'SubjectOf(m)'.txt.

Constant repetitions can be easily detected by searching for repetitions in the user's history. This approach is adopted by APE using the KMR algorithm [5]. The main limitation of APE is that it is unable to detect a repetition if the order in which the actions are performed varies from an iteration to the next. In the table formatting example above, the user may sometimes set the number of rows before setting the number of columns. Also, because of the KMR algorithm, APE is not incremental.

Detecting a **loop** in the user's behavior requires the system to identify, from a few iterations, the body of the loop and the set of objects over which the body is repeated. Eager was one of the first implicit PbE system able to detect loops in the user's behavior. APE extends Eager in that it is able to automate loops even if the iterations are not consecutive in the user's history (i.e. interleaved with non repetitive actions). In the loop example above, this occurs if the user performs some non repetitive operations on each newly modified picture before processing the next picture. Each time APE finds a constant repetition (using the KMR algorithm, as explained above) it assumes it is a loop candidate and checks if it has been iterated over objects belonging to the same set. To this end, it compares the actions immediately preceding (resp. following) the occurrences of the repetition using a similarity measure based on predefined sets like sequences of integers, days of the week, alphabetic sequences, "files in the same folder", "subclasses of the same class", etc. APE is, again, limited by the fact that the KRM algorithm is not incremental and does not detect repetitions when the order of the composing action varies. Also, both APE and Eager are limited in that they use some predefined domain dependant similarity measure to detect loops.

Using the technique described for loop automation above, and with the same limitations, APE detects and automates **nested loops**.

Conditional loops and **variable loops** are more difficult to detect. We are not aware of any implicit PbE system automating them. Clearly, the difficulty lies in the fact that the iterations are all different (no action repeats exactly) making iteration identification impossible with repetition searching algorithms.

We have shown in this section that learning what to automate in implicit PbE is based on two processes, namely identifying iteration examples and inferring a predictive pattern from these potential examples. The approaches proposed for the first process, identifying examples, have two limitations. First, they are not incremental. However, incremental mining of sequential patterns is an important subject of research in data mining, suggesting a direction for future research. Second, they are cases where repetition searching algorithms are not appropriate. This is the case, for instance, when the order in which the user has performed the actions composing them varies (action ordering problem) or for conditional and variable loops where iterations have few syntactic similarities. We believe that a promising solution consists in recording user's actions in a structured, multi-level history as proposed by Kosbie and Myers [6]. For example, Figure 1 depicts a structured representation of a high-level "Save-Mail" action which includes a "Save-As" action, which itself includes lower level actions. Such a representation clearly solves the action ordering and syntactic dissimilarity problems since they reduce to detecting the repetition of a higher level action.

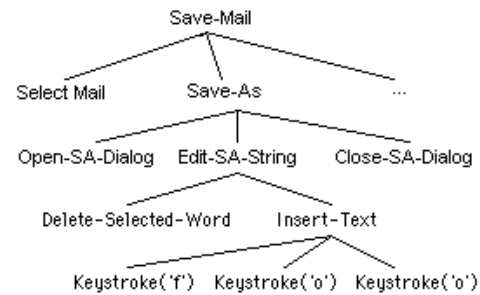


Figure 1. A multi-level representation of a "Save-Mail" action.

The approaches proposed for the second process, inferring a predictive pattern from potential examples are limited in that they rely on some predefined domain dependant operator to identify the objects over which the iterations are repeated, and to predict future iterations. Lau and Weld [7] made a significant step towards domain independent operator. They have shown that repetition automation can be seen as a prediction problem: predicting the next iterations from previous iterations. This allows PbE systems to use various machine learning algorithms to predict next iterations. However, the detected potential examples of previous iterations may not all fall in the same iterative task, or may not fall at all in an iterative task. In other words, rather than strictly trying to predict next iterations from all the examples, implicit PbE system have to be able to examine several alternative hypotheses about the iterative task at hand. Lau et al [8] proposed an interesting algorithm to achieve this goal based on a structured representation of the user behavior similar to the multi-level history described above.

3.2 LEARNING WHEN TO MAKE A SUGGESTION

Implicit PbE systems, and most modern explicit PbE systems, are built around the assistant metaphor. Once they have identified some repetitions in the user's behavior and learned how to automate them, they have to determine when to offer to the user to perform them on his behalf.

We explained in the previous section that Eager assumes that loop iterations are always consecutive in the user's history. This limitation is also a strength since it allows Eager to determine easily when to make a suggestion and what to suggest: as soon as it detects two consecutive iterations of a loop, it offers to complete the loop. Alternatively, APE does not assume consecutive repetitions. As a consequence it has to determine, after each user's action, if he is about to perform something repetitive and if it can offer some assistance. APE builds on predictive interface research and uses machine learning to model correlations between the situations the user encounters and the repetitions he performs. It afterward uses the learnt model to predict user's action and anticipate repetitions. More precisely, APE takes into account the situations in which the user performed a repetition but also the situations in which he did not. It learns two models of the user's behavior, a model to predict if the user is about to perform a repetition and, if he is, a model to predict which repetition he is more likely to perform.

As we explained, learning when to make a suggestion to the user is a typical machine learning problem and future advances will probably come from this area of research. Some authors, however,

have shown that some specificity can be taken into account to improve the performance of these algorithms. For instance, Davison et al [3] suggested to weight recent actions more highly using an exponential decay function. Yoshida et al [15] demonstrated the interest of integrating meta-knowledge (the flow of information between commands) in the representation of the user's history. Finally, we believe that meta-learning techniques like boosting or error correcting output code could advantageously benefit from the user's think time to improve the learnt models.

4. MEASURING EFFECTIVENESS

An important issue in implicit PbE is to measure the quality and effectiveness of the suggestions of an implicit PbE system. It can be evaluated using to measures from information retrieval called precision and recall. In the setting of implicit PbE, precision can be defined as the proportion of correct suggestions the system makes and recall as the proportion of repetitions for which the system makes a correct suggestion (i.e. correctly anticipated and suggested). Ruvini et Dony [13] also suggested to take into account the proportion of suggestions the system makes when no suggestion is expected. Table 1 defines precision, recall and excess according to a confusion matrix.

		The user performed		
		R	Not R	Nothing
The system predicted	R	a	b	c
	Not R	d		
	Nothing			e

$$\text{Precision} = \frac{a}{a+d} \quad \text{Recall} = \frac{a}{a+b} \quad \text{Excess} = \frac{c}{c+e}$$

Tableau 1. Definition of precision, recall and excess in implicit PbE.

We did not address in this paper the problem of suggestion readability and presentation. Clearly, they have to be as unobtrusive as possible to minimize cost of prediction errors. Also, the number of suggestions the system makes to the user has to be thought carefully. Too few suggestions limit system recall while too many suggestions put a workload on the user to browse through the list of suggestions. APE solves this problem by presenting suggestions in an independent window that the user can resize, deciding implicitly how many suggestions are presented.

5. SUMMARY

Programming by Example research focuses on making computer easier to use by alleviating users from tedious repetitions. In this paper we emphasized the distinction between explicit PbE based on the macro recorder metaphor and implicit PbE. Implicit PbE systems minimize as little as possible user's intervention and efforts. They watch over the shoulder of the user, detect repetitions in his behavior and automatically offer to automate these repetitions when appropriate.

We explained that the task of an implicit PbE system can be split in two subtasks, learning what to automate (the repetitions) and learning when to make a suggestion. We reviewed the main

technical difficulties they raise and we showed that both propose interest challenges to data mining and machine learning research.

6. REFERENCES

- [1] Armstrong, R. and Freitag, D. and Joachims, T and Mitchell, T. "Webwatcher: A Learning Apprentice for the World Wide Web". In AAAI Spring Symposium on Information Gathering, 1995.
- [2] Cypher, A. "Watch What I Do: Programming by Demonstration". MIT Press, Cambridge, MA, 1993.
- [3] Davison, B. and Hirsh, H. "Probabilistic Online Action Prediction". In Proceedings of the AAAI Spring Symposium on Intelligent Environments, 1998.
- [4] Kahn, K. "Generalizing by Removing Detail: How any Program Can Be Created by Working With Examples". In H. Lieberman editor, Your Wish is My Command: Programming by Example, Morgan Kaufmann Publishers, Boston, 2001.
- [5] Karp, R.M. and Miller, R.E. and Rosenberg, A.L. "Rapid Identification of Repeated Patterns in Strings, Trees and Arrays". In Proceedings of the Fourth Annual Symposium on Theory of Computing, ACM Press, p. 1-3, Denver, 1972.
- [6] Kosbie, D.S. and Myers, B.A. "A System-Wide Macro Facility Based on Aggregate Events: A Proposal". In A. Cypher editor, Watch What I do: Programming by Demonstration, MIT Press, London, 1993.
- [7] Lau, T. and Weld, D.S. "Programming by Demonstration: An Inductive Learning Formulation". In Proceedings of IUT'99, ACM, Redondo Beach, CA, 1999.
- [8] Lau, T. and Wolfam, S.A. and Domingos, P. and Weld, D. "Programming by Demonstration Using Version Space Algebra". Machine Learning, 53(1):111-156, 2003.
- [9] Lieberman, H. and Nardi, B. A. and Wright, D. J. "Training Agents to Recognize Text by Example". In H. Lieberman editor, Your Wish is My Command: Programming by Example, Morgan Kaufmann Publishers, Boston, 2001.
- [10] Lieberman, H. "Your Wish is My Command: Programming by Example". Morgan Kaufmann Publishers, Boston, 2001.
- [11] Maes, P. "Agents that reduce work and information overload". Communications of the ACM, Special Issue on Intelligent Agents, 37(7):31-40, 1994.
- [12] Potter, R. "Just-in-Time Programming". In A. Cypher, editor, Watch What I do: Programming by Demonstration, MIT Press, London, England, 1993.
- [13] Ruvini, J.D. and Dony, C. "Learning User's Habits to Automate Repetitive Tasks". In H. Lieberman editor, Your Wish is My Command: Programming by Example, Morgan Kaufmann Publishers, Boston, 2001.
- [14] Ruvini, J.D. "Do Users Tolerate Errors From Their Assistant? Experiments with an E-mail Classifier". In Proceedings of IUT'02, ACM, San Francisco, pp. 216-217, 2002.
- [15] Yoshida, K. and Motoda, H. "Automated User Modeling for Intelligent Interface". International Journal of Human Computer Interaction, 3(8):237-258, 1996.