Just-in-Time Knowledge for Effective Hybrid Learning

Michel C. Desmarais

Computer and Software Engineering Department
Polytechnique Montréal
michel.desmarais@polymtl.ca

Abstract. The means for Hybrid learning take on many forms. In this paper, we look at learning facilitators that can be embedded within the user interface. We argue that these means of learning can be even more effective than formal training. We describe different features of the user interface that can provide just-in-time knowledge that fosters learning: immersing the student into a rich environment where he can readily have access to the information for the task at hand.

1 Introduction

The European Commission is devoting over 50M Euros annually in recent years to fund projects that are aimed at developing new means of learning, and new means of creating and managing digital content[1]. Few of the funded research, if any, aim to develop technologies intended for the classroom. Instead, the emphasis is on developing technologies to make the delivery of learning content more individualized, interactive, and embedded into our everyday environment.

2 Learning Occurs Naturally, but It Can Use Facilitators

Why would such means of learning have a great potential of being effective?

Looking at the learning phenomena in general, we know that most of what we learn occurs outside of a structured context such as a classroom. Any researcher in Artificial Intelligence can attest that most cognitive tasks that humans perform involve a phenomenal amount of knowledge that was acquired throughout life. Much of this knowledge has to do with problem solving skills and "common sense" inference, which is mostly acquired in a semistructured or non structured environment, through practice. This huge amount of knowledge is, needless to say, not found in textbooks.

Language is a good example of our ability to learn in an unstructured context. Learning a language starts with the imperative need to communicate, and with the environment in which that language is omnipresent. The combination of *need* plus *environment* is sufficient to incur the learning of a complex skill.

[1] http://cordis.europa.eu/fp7/ict/telearn-digicult/telearn_en.html (last consulted on 2008.05.12).

J. Fong, R. Kwan, and F.L. Wang (Eds.): ICHL 2008, LNCS 5169, pp. 31–39, 2008.

Now, this is not to say that we should do away with structured learning and that unstructured learning is the only and best way to learn a subject mater. The point is that when we get the co-occurrence of the *need to know* something, or the *need to perform* some task, and an environment that *provides the elements to learn and perform*, then learning will naturally occur. What makes unstructured learning so powerful, is that learners often have a constant need to know or to perform. It is up to us to provide them with the proper environment that can foster that learning. The constant availability of that environment and the prevalence of the need to know and perform can far outweigh the time and the attention the learner can devote to learning in a structured context, such as a classroom.

The question is, how can we best leverage over this type of learning in the context of hybrid learning? A key factor is the constant progression, in recent decades, of the computerization of our environment, and of our access to knowledge and information that has dramatically increased. That opens an opportunity to enrich our environment with the right features that can foster unstructured learning towards specific learning objectives, and complement the more traditional structured learning. Learning through our computerized environment is already well under way, and our purpose here is to reflect on the most the most exiting developments.

3 Environments That Support Learning

Let us take that idea of fostering autonomous learning and look at the kinds of means we deploy to enrich the learner's environment towards that goal. We focus on the transformation of standard user interfaces that were originally designed with the goal of providing functionality, **to interfaces that embed the goal of inducing the learning of complex task**.

We start with an example that we had the opportunity to work on, the THEO Electronic Performance Support System (EPSS).

3.1 The THEO Example

THEO is a system designed for the customer support centers at a large utility company with millions of clients. The original interface is displayed in Figure 1.

The tasks that the user can perform with THEO are quite diversified and sometimes non trivial, such as explaining how a fixed monthly payment is derived from year to year, or making a diagnostic of a sudden increase of electricity consumption in a household. As a consequence, the classroom training for a new employee required three weeks.

In order to reduce the training period, we integrated a number of additions to the original THEO interface that are meant to provide a kind of just-in-time training, often referred to as an Electronic Performance Support System (EPSS) [1]. These features are accessible from the original interface in Figure 1 and are not intended to replace it. Their intent is to allow learning through the user interface.

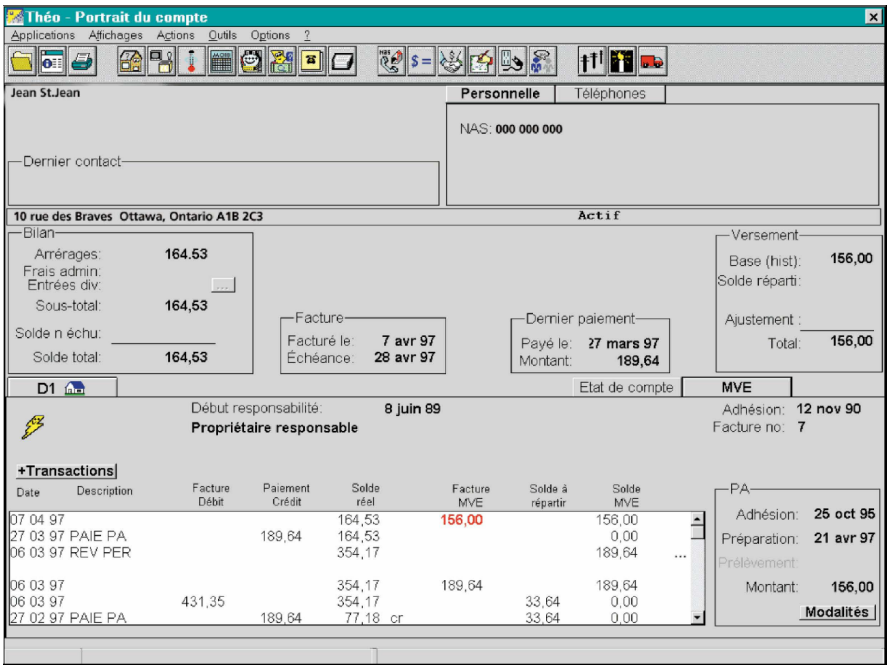


Fig. 1. Original THEO interface

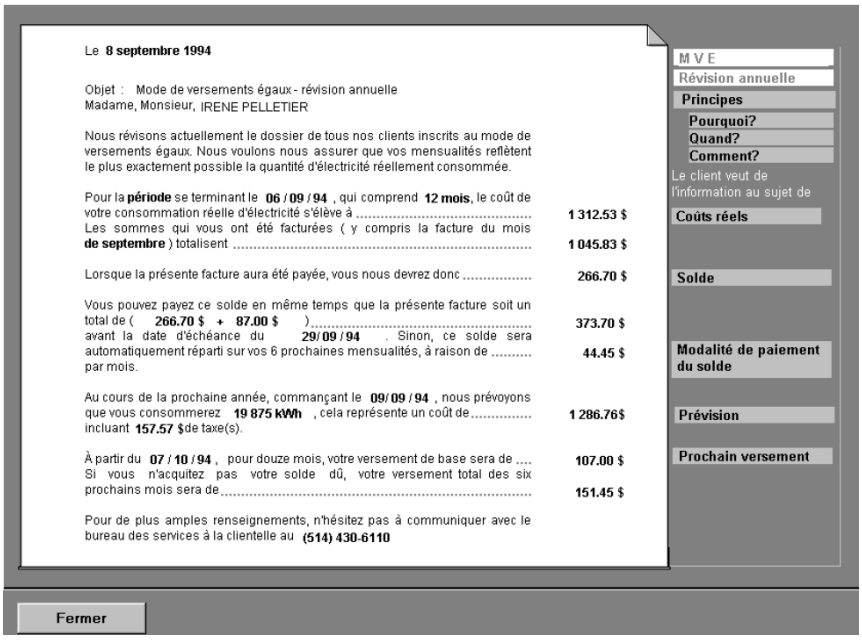


Fig. 2. Access to the letter sent to the client

Letter
The first and simplest mean of support consists in providing access to the most recent letter sent to the client by the company where the numbers are highlighted (Figure 2). This is in general what the client has in hand when he calls the CSR (Client Support Representative). It gives the CSR a common point of reference in the exchanges with the client.

Hyperlink
The second means consists in making expandable each of the highlighted number in that letter, akin to hyperlinks (Figure 3). As most questions refer to these numbers, that mechanism allows easy, access to the details of how each amount on the letter is

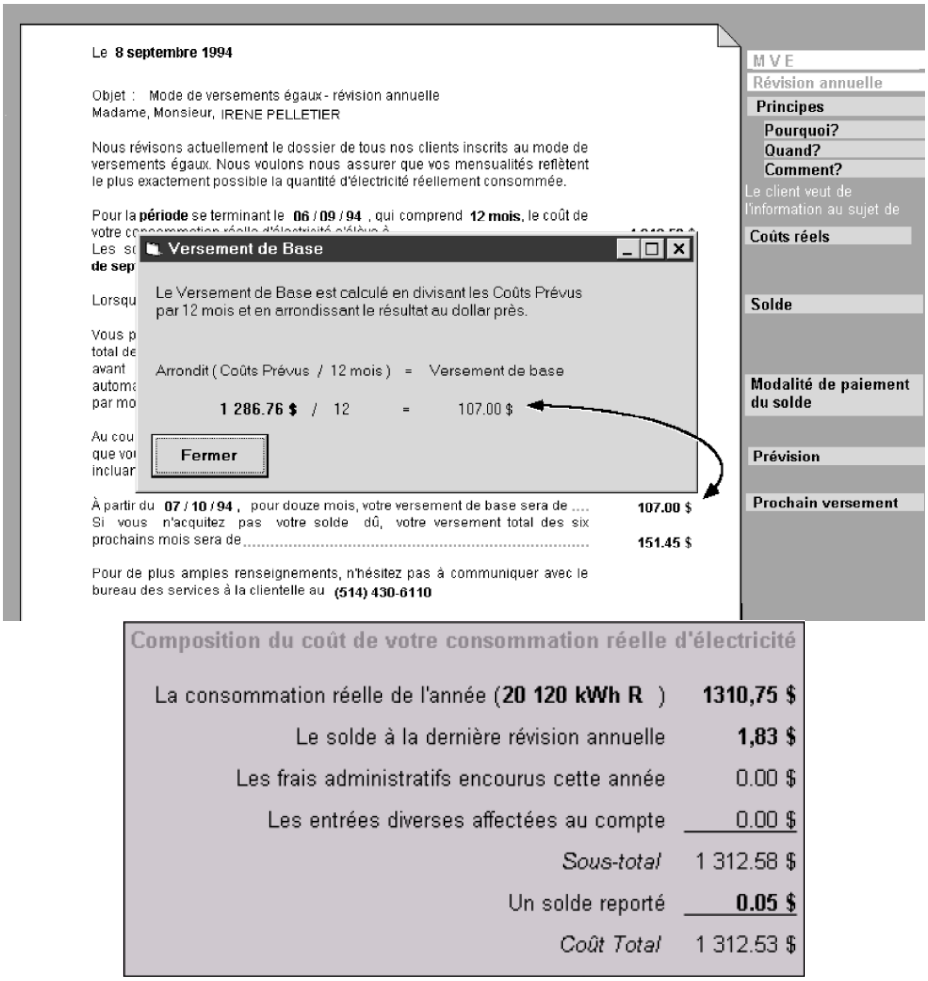


Fig. 3. Two examples of hypertext-like expansions of amounts that provide explanation on how they are derived

derived. The user clicks on a number and a window shows how this number is derived. The numbers in the explanation window can be further explored this way.

Diagnostic and documentation

Another feature of the interface is its ability to facilitate access to the relevant information by inferring the most likely causes of a CSR call. It allows direct access to the part of the system that is needed to answer the client's question. For example, Figure 4 shows the three most likely causes of a CSR call based on a statistical analysis of the client's profile. If the reason for the call is, say, the first item shown (a 26% increase of annual electricity consumption), the plausible causes are reminded to the CSR and can be discussed with the client. When applicable, all numbers shown are computed to reflect the actual impact in dollars for that particular client to make the information more relevant.

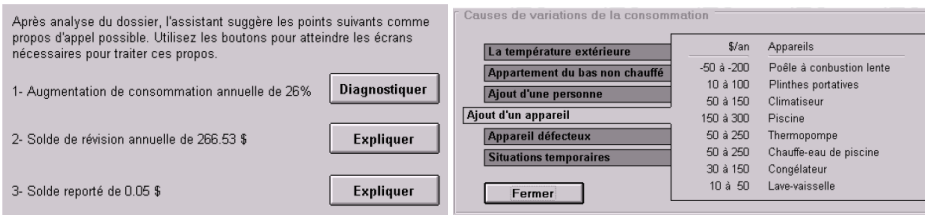


Fig. 4. Inference of the three most likely causes of a CSR call (left) and a corresponding explanation of cause 1 (right). Each tabbed text display in the explanation windows highlights, for that particular customer, the amount that corresponds to the different causes.

Evaluation Results

The impact of introducing the EPSS to the original THEO interface were investigated in an informal experiment (see [1] for details). The results of this experiment show that two out of three users with no training at all were able to perform the standard 15 out of 15 tasks relating to the topic chosen for the experiment (the equal payment plan that normally requires three weeks training). The other subject was able to perform 10 out of the 15 task. In comparison with the original interface without the EPSS, only one of the tasks was succeeded by a person without training. However, the trained CSR representatives were able to do almost all of them, as expected.

These results form a compelling argument for the effectiveness of the EPSS enhanced user interface. Considering that two out of three subjects were able to complete all of the 15 CSR tasks without training, it is quite reasonable to consider that the original three weeks training could be substantially reduced given the EPSS.

A qualitative questionnaire was also administered to investigate non performance related factors. Results from this questionnaire were very positive. All subjects, whether novices or experts, had positive comments on the EPSS. They all considered this tool to be useful, especially for novices. The number inspection technique was preferred over the hierarchical decomposition. It was unanimously considered "very useful" by all five of the subjects who filled out the questionnaire, whereas the hierarchical decomposition technique was considered by two subjects as "very useful" and by the three others as "rather useful".

The authors concluded from this investigation that an EPSS like THEO can substantially reduce the training period, and even improve the quality of the CSR service in general. The investigation clearly showed the power of rethinking the user interface to extend its purpose beyond the sole goal of providing the functionality, to the goal that encompasses the training and the learning of the most complex tasks to perform with the system.

3.2 Interactive Development Environment Examples

The power of embedding, within the user interface, means to help the user learn complex tasks has now been recognized within some communities. One of the most notable example is in the computer programming community where IDE (Interactive Development Environment) have evolved into highly sophisticated interfaces to support computer program development. These environments allow the programmer to have access to a large array of contextual tips, documentation, and other interface features that help them not only in doing the task more efficiently, but also to better learn the programming language and more advanced programming techniques.

There are many examples that could be mentioned here, and we name but just a few for brevity.

Project template skeletons and examples
The typical software project development types (GUI, library, etc.) are provided as template code that is complete with default structure and configuration, relieving the programmer from the initial effort of finding sample code to start from, and providing a useful example for the novice programmer.

Syntax
When typing, all syntactically incorrect lines are highlighted and, when possible, the correction is suggested.

Auto-completion
A powerful feature is the ability of the IDE to analyse the code and display a list of possible completions to the expressions as the user is typing them, such as the list of all methods that can be called upon a given object. Not only does this feature relieve the load on the user's memory, it also allows him to explore and learn the possibilities of the library.

Source code
All of Java's source code base is available by clicking over the corresponding function call in the user's program. Exploring the inner architecture of Java's source code helps the programmer understand the framework and learn advance programming tips and patterns.

The result of these features is twofold :

1. the user has fewer things to know, and thus fewer things to learn;
2. the learning occurs naturally as the user performs the tasks.

The case of *auto-completion* is a clear example of how this result can occur (Figure 5). It is taken from the NetBeans IDE[2]. The two popup windows of the NetBeans interface screendump show the auto-completion feature in action (refer to the two popup windows pointed to by the "completion and documentation" bubble text in Figure 5). The user is typing the name of the "System" class and the top window shows the applicable methods for that class, whereas the bottom window shows some details about the highlighted choice.

In this example, the cognitive load of remembering the name of the methods is reduced to a task of recognition and, in the case where the method is actually unknown, it provides immediate documentation to find and learn the method that should be used for the intended goal.

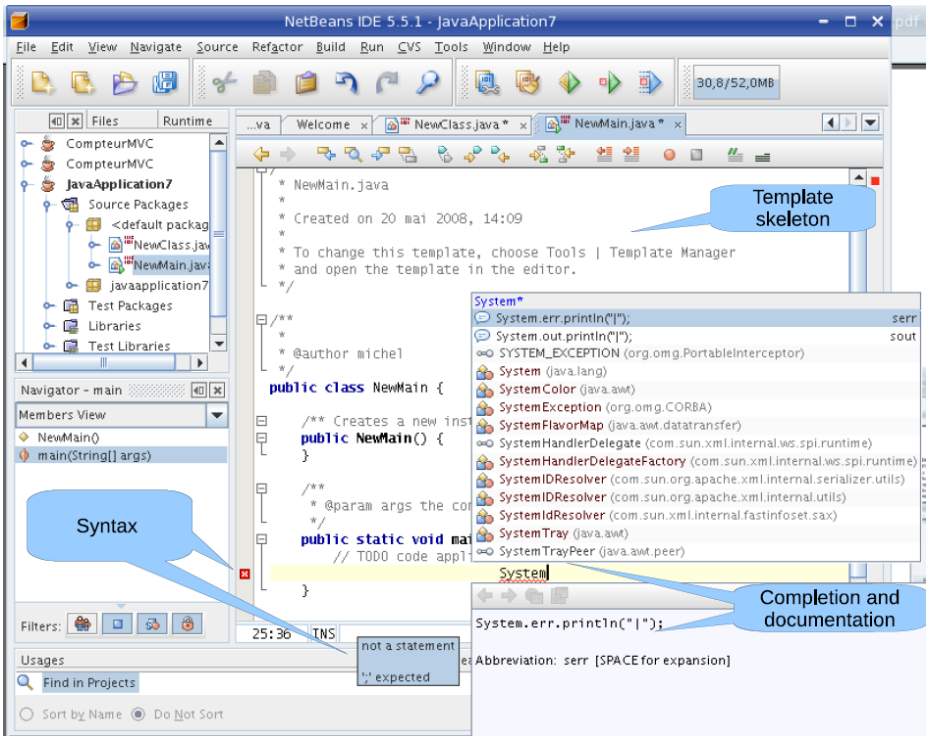


Fig. 5. Some features of the NetBeans IDE interface such as Auto-completion as when the user is typing the name of the class "System", syntax correction, and code template skeleton

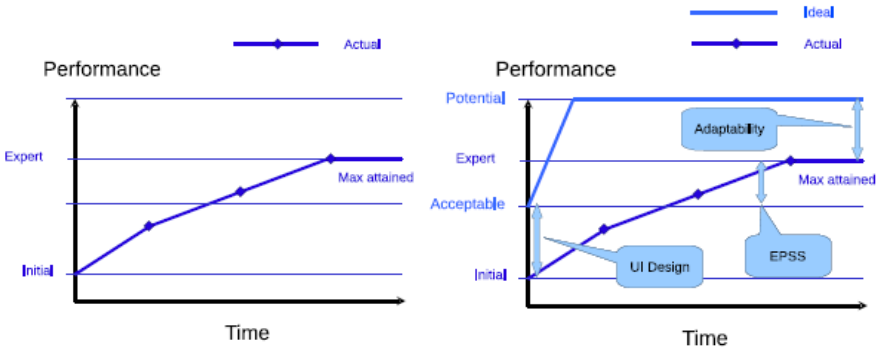
All the other features mentioned above serve the same purposes of reducing the amount of learning, providing examples, highlighting errors at the very moment the user makes any and of providing highly precise and context sensitive documentation to learn to do a task.

[2] See http://www.netbeans.org/. The same could be said of most popular IDEs such as Microsoft's .NET, Borland's Turbo series, or IBM's Eclipse.

4 Interface Design, Learning, and Performance

The observations from the THEO and NetBeans interfaces bring us to a larger understanding of the interactions between interface design, learning, and performance. These interactions are illustrated in Figure 6.

The learning curve of Figure 6(a) is typical of the evolution of performance. It follows the well known Power law of practice [2]. As the user gets familiar with the tasks and the application, he develops strategies to perform better. The performance increases from an initially low level to an expert level and then levels off. However, this curve is not necessarily the best that can be achieved. Often, interface designers settle for a design that allows the user to perform at an acceptable level quickly, but that design may not be optimal for expert users. Moreover, expert users tend to stop their learning process too early because they lack the incentive to make the effort of consulting the necessary documentation, or the time to explore new and complex functionality that could him to new levels of performance.



(a) The learning curve (b) Support Means

Fig. 6. The learning curve and interface design issues

Figure 6(b) depicts the "Ideal" curve. Long term performance reaches a greater level of performance and the initial performance also starts off at a higher level. This goal is difficult to reach because there often is a tradeoff between initial performance and the potential performance that can be reached with an interface. Some interfaces can have a steep learning curve but, once the user has gotten over the initial effort of mastery, the performance can be much greater than with a simpler interface to start with. On the contrary, interfaces that are very easy to use for novice users often do not meet the needs of high performance for expert users.

Three means can be deployed to avoid this tradeoff between initial ease of use and the optimal performance for experts, and, thus, to transform the "actual" performance curve into the "ideal" one:

UI Design

The first one is the best known and consists in good User Interface (UI) design. Proponents of the user-centred approach to developing interactive software know that a

good UI design can make a substantial difference in the user productivity. Studies have shown that a gain of 35% in productivity is what can be expected [3] by properly applying a user-centred approach to UI design. One of the key element of the design here is to support the diversity of users, namely experts as well as novices.

EPSS
The second means consists in using Electronic Performance Support Systems (EPSS). This is what is depicted in the THEO and NetBeans examples. As argued above, it enriches the user environment with embedded mechanisms to foster learning in a just-intime, context sensitive and on-demand fashion. The user can learn to gradually perform more complex tasks, more efficiently, in a naturally occurring manner that is akin to how most of our learning is acquired.

Adaptability
Finally, the third mean refers to the ability of an interface to adapt to its user. For the purpose of training and learning, one of the most important adaptation is the ability to adapt the interface help, documentation, and guidance to what the user knows and his level of skills for a given task. Another is the ability to infer what is the current user's goal. The auto-completion is such an adaptation feature. This feature is the most difficult to implement and the advances in the field are slow, in spite of substantial research efforts in the field during the last two decades[3]. However, in the long term, it should yield significant returns.

5 Conclusion

Our daily environment is becoming more computerized than ever before. We argue that we need to think of the user interface of our computerized environment not only as a tool to perform tasks, but as a tool to help increase our skills and learn through that very interface. We have outlined different means of doing so and shown that they can prove very effective towards that goal.

At a time where the constructivist approach is paramount and where learning by doing is perceived as a key element for most programs in schools, the potential of reaping the benefits of enriching our computerized environments with learning facilitators is great. The opportunity for teachers to leverage on such learner centered environments is something to consider.

References

1. Desmarais, M.C., et al.: Cost-justifying electronic performance support systems. Communications of the ACM 40(7), 39–48 (1997)
2. Newell, A., Rosenbloom, P.S.: Mechanisms of skill acquisition and the law of practice. In: Anderson, J.R. (ed.) Cognitive Skills and their acquisition, pp. 1–55. Erlbaum Associates, Hillsdale N.J (1981)
3. Landauer, T.K.: The Trouble with Computers. MIT Press, Cambridge (1995)

[3] See, for example, the User Modeling and User Adaptive Interfaces Journal.