

Making Useful Programming Objects Available for a Programmer: at the Right Time in the Right Way through the Right Peers

Yunwen Ye¹

¹Center for LifeLong Learning and Design

Department of Computer Science

University of Colorado

Boulder, CO80309-0430

+1 303 492 3547

yunwen@cs.colorado.edu

Kazuaki Yamada²

²Knowledge Interaction Design Laboratory

RCAST, University of Tokyo

4-6-1 Komaba, Meguro

Tokyo, 153-8904, Japan

+81-3-5452-5286

kumiyo@kid.rcast.u-tokyo.ac.jp

Kumiyo Nakakoji²

Abstract

Programming is a complex ill-defined problem-solving task. It requires not only knowledge in the head of a programmer, but also existing information in the world and relevant knowledge of his/her peers in the community. Traditional software reuse techniques do not serve for programmers to a full extent because they are not built from this perspective. Our research addresses the issue from a cognitive perspective, trying to help a programmer reuse existing software objects from software repositories and locate peer programmers who can help him/her solving the current task even when (1) the programmer is not aware of his/her information needs, (2) does not know that potentially useful information exist, and (3) does not know how to ask for help in related programming communities. This paper discusses the design principle called *layered information-on-demand* that guides the location and presentation of external information and knowledge in response to an individual's unique information needs determined by his/her current task and background knowledge. The principle is illustrated with the CodeBroker system and the UCM (user-modeling and community-modeling) engine.

1. Introduction

More and more computer systems are designed to support complex tasks, such as programming, city planning, and design. Few users have all the knowledge required to accomplish these complex tasks; they have to

rely heavily on external information and on knowledge of other people. The effective and efficient accomplishment of complex tasks is therefore no longer confined to an individual user but has to rely on distributed cognition [22] by reaching into a complex networked world of information and through computer mediated interactions [12]. Timely and effortless access to external information becomes crucially important for users who are engaged in solving complex tasks with computational tools.

Software reuse can be regarded as an approach to help a software programmer make use of external information. Many of previous reuse research projects, however, have primarily focused only on designing various searching or browsing mechanisms to assist software developers in locating components [16]. Both browsing and searching are *passive* mechanisms because they become useful only when software developers decide to make a reuse attempt by knowing or anticipating the existence of certain components. They are of little use for less experienced software developers who do not even anticipate the existence of components or do not know how to search the reuse repository properly [28][29].

As Poulin [20] claims, modern reuse repositories contain not only 30 to 250 components, but also thousands of components; for example, the Java 1.4 API library has 2,723 classes. It is therefore becoming increasingly difficult for software programmers to know or anticipate the existence of all available components.

To enable the reuse of components whose existence software developers do not even anticipate and therefore cannot locate with traditional searching and browsing mechanisms, we have built a design principle called *layered information-on-demand*. The principle guides the

design of interaction and coordination between users and external resources, which include software objects from reuse repositories and knowledge of skilled coworkers. It aims at providing a natural link form the task at hand to a variety of external information and knowledge resources, which are presented to users at different levels of abstraction in response to their different needs.

Depending on the working context and background knowledge of a user, a unique network of information and knowledge is dynamically constructed and presented to provide a seamless transition from users' interaction with tools to interaction with external information and other knowledgeable peers.

The design principle of *layered information-on-demand* supports the two kinds of distribution of cognitive process identified by Hollan et. al. [12]: (1) cognitive processes that involve the coordination between internal and external structure, and (2) cognitive processes that are distributed across the members of a social group.

The paper is organized as follows. After discussing the issues to be considered for the principle of *layered information-on-demand*, we present its instantiation in a programming environment. Related work and the possible application of the principle to two widely used systems are discussed before the summary.

2. The Principle of Layered Information-on-Demand

2.1 Problems being Addressed

The advancement of information technology has created a huge wealth of information based on the vision of “anywhere and anytime.” The scarcest resource is not information anymore; it is *attention*. The fundamental challenge now is to help users to “attend to the information that is the most useful or interesting or, by whatever criteria you use, the most valuable information [24].”

The drastic imbalance between the huge amount of information available and the limited attention of human beings [3] represents one of the most fundamental research challenges. Figure 1 depicts four different levels a user typically has about an information system [27]. Rectangle L4 represents the actual information system, and the ovals (L1, L2, and L3) represent a particular user's different levels of knowledge about the system. L1 contains the elements that are well known and can be easily used. L2 contains the elements the user knows vaguely and uses occasionally. L3 contains the elements the user anticipates to exist. The cloud contains the elements needed for the task at hand. The black dots are not relevant to the current task while the white dots represents elements that are already known to the user.

The essential challenge for computationally supported environments is how to locate the task-relevant

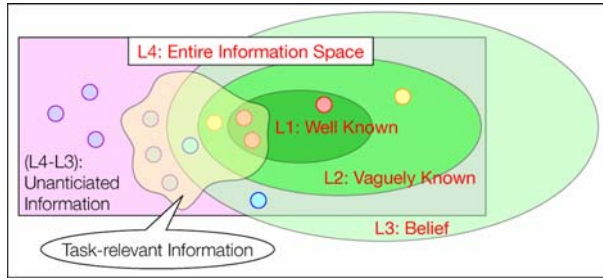


Figure 1: The Challenge: Finding the Task-Relevant Information a User Does Not Know

information a user does not know (the shaded dots), and present it to the user in a way that is natural to their work practice and cognitive process.

Inoue et al. [13] has been addressing the challenge of locating software objects relevant to the current task by ranking software components. Their Component Rank system is based on the analyses of actual use relations among the components and propagating the significance through the use relations. Their approach, however, does not solve the problem when users:

- are not aware of their information needs;
- do not know the existence of useful external information;
- are not able to understand the newly found objects and how to apply them to the current situation; and
- do not know whom they can ask for help.

Our approach is a complement to the work by Inoue et al. where the goal of our research is to provide relevant information *at the right time in the right way through the right peers* for a programmer. This approach can be contrasted with “anywhere and anytime” approach.

Layered information-on-demand supports a new interaction style between information systems and users that focuses on the economical utilization of human attention. It is a type of progressive information presentation. It exploits distributed cognition in the following ways:

1. Information is provided on demand, namely when the information is needed by users for the accomplishment of their task [8].
2. Information is personalized to the background knowledge of a user so that each user is given only the information he or she does not have in head [10].
3. Information is directly presented into the working environment of users, who can use the information as an augmentation to their own knowledge [27].
4. Information is presented at different levels of details to accommodate the individual difference of users in evaluating, understanding, and applying information [23].
5. Information is used to structure the collaboration among users [11].

The following subsection details each aspect.

2.2 Elements of the Principle

2.2.1 Information on Demand

Information-on-demand can be easily achieved when users know where and how to find the information by browsing or searching. Information-on-demand becomes challenging when users are not aware of the existence of information that they can take advantage of (i.e., the elements in L4–L3 in Figure 1) [1], or do not know how to find the needed information. Information systems have to infer high-level information demands from low-level user activities in a workspace [19] and then proactively *deliver* the needed information [25]. Plan recognition [4] and similarity analysis [10] are two major approaches to inferring demands.

2.2.2 Personalized Information

Different users have different knowledge. Even for the same user, their needs for information change when they learn. To provide personalized support for each user, the system needs to know what the user has known already. User modeling is an approach to this goal [10]. To reflect the increase of user knowledge, user models need to be both adaptable and adaptive. Adaptable user models allow users to update their user models directly. Adaptive user models require the system automatically updates user models by learning from the interaction between users and systems.

2.2.3 Knowledge Augmentation

Cognitive activities involve a rich interaction between internal resources (e.g., attention, knowledge) and the surrounding external resources (e.g., objects, information) [12]. Cognitive activities are affected by both the internal knowledge in the head and the external information present in the working environment [24]. When information is directly accessible from users' current working environment, it can actively participate in the cognitive process of users, in a way similar as the knowledge in their head does. In this sense, immediately accessible information augments the knowledge of users.

2.2.4 Details on Demand

Locating, comprehending, and using information involves at least three different phases: information discernment, detailed evaluation, and information application [5]. In the *information discernment* phase, users quickly determine whether a piece of information is relevant to their current task. The *detailed evaluation* phase involves a relatively thorough study of the

information. In the *information application* phase, users need not only to understand the information but also to apply it in their current situation.

Each phase requires different levels of details of the information. For the information discernment phase, an overview of the information is enough. This phase also requires the simultaneous presentation of several candidates for users to compare and find the information that is most relevant. For the detailed evaluation phase, users need to focus on the details of a chosen piece of information. Examples that use the information are very effective in helping users understand, adapt, and integrate the information by providing context and enabling users to draw an analogy between their task and examples [9].

The details required for information comprehension and use also depend on the knowledge of users. A continuum of demands for different levels of details exists. On one extreme, if the user already knows the presented information vaguely (L2 in Figure 1), an overview of the information may be enough. On the other extreme, if the user has never encountered the information before (L3 and L4–L3 in Figure 1), he or she may need to go through all the three phases, and that iteratively because he or she may not be able to make the right choice at first.

Presenting information in different levels of details and on demand from overviews to details to examples is an important way to satisfying the different needs in each of the three phases, and to accommodating the differing needs of each individual user by putting control at users, who can stop at any desired level whenever they deem the presented information enough.

2.2.5 Mediating Collaboration

Skilled practitioners are efficient because they are able not only to utilize their own knowledge and external information but to seek the help of other knowledgeable peers [2]. Merely providing information is not enough. Information systems should also facilitate the access to the expertise residing in other knowledgeable peers by mediating collaboration [6]. Expertise is not an attribute of a person; it is an attribute of both the task and the person. A basic challenge to mediate peer collaboration is to identify the experts for a particular task, and to find an expert who is able to and willing to help at a particular moment because experts have their own work and do not want to be interrupted too often.

3. Supporting Programmers with Layered Information-on-Demand

In this section, we use CodeBroker system [27] and the UmCm (User-Modeling and Community-Modeling) engine [18] as prototyped systems to illustrate how layered information-on-demand is implemented to support Java programmers. CodeBroker delivers

information on reusable components to Java programmers on demand and presents the information in different layers that range from a synopsis right in the current programming environment to an expert who is willing to help. The UmCm engine identifies which community a user should ask for help by identifying a community model based on the user's past interaction histories.

In object-oriented programming languages, such as Java, the knowledge of reusable components (classes and methods) in its class libraries is essential for better programming quality and productivity. The huge number of reusable components poses great learning challenges for Java programmers. Few Java programmers, if any, know all class libraries. Currently, programmers have to either learn those components in advance or browse through the library documentation system to find components that are needed for their current task. This is not only time-consuming but very ineffective. Programmers may not start searching for the component at all because they do not even know it exists (i.e. L4-L3 in Figure 1). Even if they want to, they may not be able to find it in the network of class libraries. Even if they are able to find it, they may not know how to use it.

CodeBroker aims to radically change the way that programmers use and learn class libraries by realizing the principle of layered information-on-demand. It is integrated with a program editor (Emacs) to provide direct access to unknown reusable components from the programming environment through its information delivery mechanism. We use a scenario to illustrate how CodeBroker helps programmers program with external information and knowledge about reusable components.

3.1 A Scenario of Using CodeBroker

This scenario is adapted from an evaluation experiment with a subject (we call him Jack) who was an experienced Java programmer.

Jack is asked to implement the following task:

Traditionally, the Japanese write numbers with a comma inserted at each fourth number from the right. For example, 1,000,000 is written as 100,0000. Implement a program that transforms the Chinese writing format (100,0000) to the Western format (1,000,000).

This task can be implemented in several different ways by using different components in the standard Java API library. The simplest solution is to use the `format` method from the `NumberFormat` class in the `java.text` package. Not even knowing the existence of the `java.text` package, Jack plans to write a program "to parse the number, take out the commas and insert the commas."

3.1.1 Autonomous Location

As usual, Jack starts with defining the interfaces of all the methods he needs. As soon as he enters the back slash in the editor that ends the document comment (where the cursor is located in Figure 2a), CodeBroker extracts the comment and uses it as a query. A list of task-relevant components that matches the query is autonomously retrieved from the Java API library.

3.1.2 Personalization

The retrieved task-relevant components are compared against Jack's user model that contains the components he already knows. The known components are removed from the list to personalize the retrieval results to Jack, because Jack would be able to reuse those known components by himself if they are reusable in the current situation.

3.1.3 Delivery

The personalized task-relevant components are delivered into the `RCI-display` buffer (Figure 2b). Each delivered component has the rank of the relevance between the component and the current task, the relevance value, the name, and the synopsis of its functionality. This is a context-aware list of reusable components for Jack to browse and serves as the first layer of information presented on demand. Not all the components are useful for the current task, but many of them are. In fact, traditional recall and precision measurement shows that for achieving 80% recall rate, the precision rate is 33.71%. In other words, about 1/3 components returned could be used to implement the task in one way or another [27].

3.1.4 Mouse movement-triggered information

Looking at the names and synopses of the delivered components, Jack realizes those with the name `format` could be used in his program, but he is not sure which one he should use. He moves the pointer over the component names in the delivery buffer (Figure 2b), which triggers the display of the component's full signature in the mini-buffer (Figure 2c), the second layer of information presented on demand to assist Jack in determining the relevance of the component. The full signature shows the package, the class, and the types of input and output data. After comparing the signatures of all the components named `format`, Jack finds the 9th component (highlighted in Figure 2b) is mostly close to what he wants because it converts a number to a string according to its signature.

To note that all the interaction with CodeBroker by far was conducted within the programming environment. This makes the location of reusable components a natural

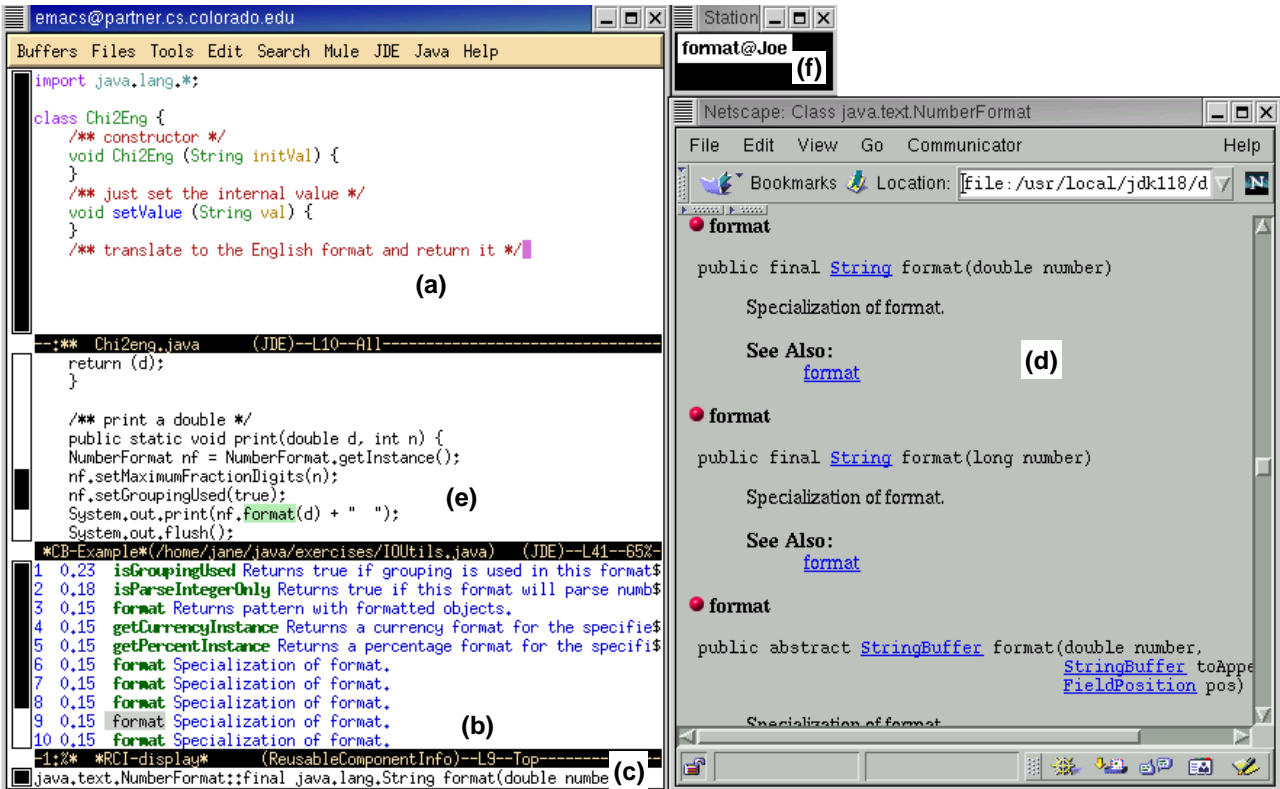


Figure 2: The CodeBroker System

extension to Jack's current programming practice because Jack does not need to make an conscientious decision to shift from programming mode to reusing mode.

3.1.5 Jumping to the documentation

Because the 9th delivered component is the most promising one, Jack decides to get more detailed information about it. Clicking on the component name brings a web browser that shows its full documentation (Figure 2d).

3.1.6 Finding an example

The documentation does not say too much. Jack still does not know how to use it. He thinks an example might be helpful. He goes back to the editor and shift-clicks the 9th delivered component to ask the system to find an example. A program that uses `java.text.NumberFormat.format` is shown in the example buffer (Figure 2e). This example buffer does not exist before Jack tries to find an example and can be removed easily after Jack is done with it. We show it in Figure 2 throughout the scenario because we want to save space by avoiding using another screenshot.

3.1.7 Finding experts

Because this package is completely new to Jack, he still has some questions. He issues the command `cb-get-expert-list`, which creates a list of experts who have used the method `java.text.NumberFormat.format` before.

3.2 Current Implementations

The CodeBroker system is implemented as a software agent that infers the demand for information by monitoring programmers' low-level activities and interaction with the system and reacts autonomously by delivering different levels of information. The UmCm engine uses social filtering mechanisms to identify a sub-community whose members share a similar interaction history with the user.

Figure 3 illustrates how the *layered information-on-demand* principle is implemented by CodeBroker and UmCm in the above scenario.

The initial delivery of task-relevant and personalized components (Figure 2b) is triggered by entering a document comment in the editing space (Figure 2a). Because the content of the document comment is often a good indication of the functionality of the following code [25], it is used as the query for reusable components. Components in the Java API class library are determined

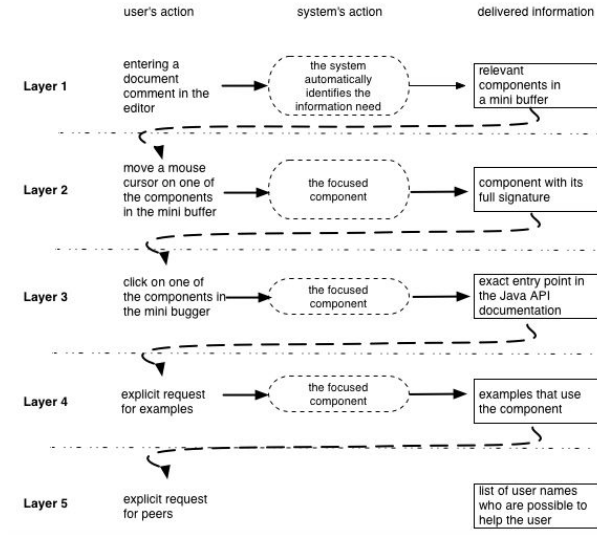


Figure 3: Layered Information-on-Demand in the Scenario

to be task-relevant and returned as retrieval results if their documents, which are also created from document comments, are similar to the inferred query based on the Latent Semantic Analysis (LSA) technique [7].

Personalization is achieved with user models [10]. An initial user model is created by analyzing the Java programs written by the programmer. Components that have been used several times by the programmer are deemed known to the programmer and are included in the user model. Before delivering components, CodeBroker removes the components contained in the user model. The user model is adaptive because when the system detects that a component is used by the programmer in a new program, it adds the component to the user model. The user model is also adaptable because the programmer can explicitly update the user model by adding known components.

The delivery of the second layer of information is triggered by mouse movement. When the programmer places the pointer over the component name in the delivery buffer (Figure 2b), the system infers that the programmer needs more information on that component and displays its full signature in the mini-buffer (Figure 2c).

The third layer of information is presented when the programmer clicks a delivered component. Each delivered component has a link, which is hidden from

programmers, to the exact place of the Java API documentation.

The fourth layer of information is displayed upon the explicit request (an extended Emacs command) of the programmer who needs an example. The editing space is split into two buffers, with the top one remaining to be the editing space and the bottom one showing an example program that uses the interested component. Examples are located from a predetermined list of directories where coworkers store Java programs. When several examples are available, the simplest one is chosen. The place where the interested component is used is highlighted to direct the programmer's attention.

The fifth layer of information, a list of experts for the interested component, is triggered by an extended Emacs command. The list is created by extracting the author names of the examples found, and then the UmCm engine formulated a group of users who have potential to be able to help the current user (Figure 4).

4. Initial Evaluation of CodeBroker

To investigate how CodeBroker assists programmers in programming with unknown components, we conducted 12 experiments with five subjects, whose expertise in Java ranged from medium to expert level. In each experiment, the subject was asked to implement a task with CodeBroker [27]. Based on the subject's current knowledge about Java library, which we obtained by analyzing the subject's recently written Java programs, we assigned a task that could be implemented either easily with some components in the library the subject had not yet known, or less straightforwardly with the subject's current knowledge.

Table 1: Overall results of the evaluation

Subject	Experiment no.	Total no. of distinct components reused	No. of distinct components reused from deliveries	Breakdown of reused components from deliveries		Rating on usefulness of the system (1: worst 10: best)
				No. of components whose existence was unanticipated (L4 - L3)	No. of unknown components whose existence was anticipated (L2, L3)	
S1	1	10	4	2	2	7
	2	3	1	1	0	
S2	3	7	1	1	0	4
	4	4	1	1	0	
	5	5	3	0	3	
S3	6	5	2	1	1	8.5
	7	4	3	1	2	
	8	3	0	0	0	
S4	9	4	3	0	3	7
	10	3	1	1	0	
S5	11	4	1	1	0	8
	12	5	0	0	0	
Sum		57	20	9	11	

The overall quantitative results are summarized in Table 1. The 12 programs created by the subjects used 57 distinct components, 20 of which were delivered by CodeBroker. Of the 20 components, the subjects did not anticipate the existence of 9 (col. 5). In other words, those 9 components could not have been used without the support of CodeBroker, and the subjects would have created their own solutions instead, as two subjects commented in interviews:

“I would have never looked up the `roll` function by myself; I would have done a lot of stuff by hand. Just because it showed up in the list, I saw the `Calendar` provided the `roll` feature that allowed me to do the task.”

“I did not know the `isDigit` thing. I would have wasted time to design that thing.”

Although the subjects anticipated the existence of the other 11 components (col. 6), they had never used them before, knowing neither the names nor the functionality. They might have used the 11 components if they could manage to locate them by themselves. In interviews, subjects acknowledged that the immediate access to task-relevant components enabled by CodeBroker’s delivery mechanism made the locating much easier and faster.

“It beats browsing. Because the way that I normally would have done the task, I would do a lot of browsing and then write the code alongside. So this reduced the browsing and searching.”

“I did not have to start browsing and go through the packages, and I did not have to go through the index of methods. I could just go to the short list [in the delivery buffer], find it and click it.”

The layered presentation of information on demand facilitated the selection of the most relevant component. We observed that subjects invariably moved pointer up and down in the delivery buffer to find the most suitable component by comparing signatures before they jump to the documentation system.

“The key benefit of [CodeBroker] is that it gives you methods for every class, not like [the Java documentation system] that you have to first find which class it is in and then go to the class. Although it has index of methods, it is hard to find.”

Some subjects stopped at the delivery buffer without further exploration after they found the information therein is enough, as evidenced in the following remark.

“I thought there might be a `parse` method, but I also was not sure whether it is called `parse` or something else. I also wasn’t sure if it was in the `Format` class. Maybe it is in a different class like `Integer` or `number` or something else. It’s helpful that I saw `parse` [in the delivery buffer] and went through to see it was in the `Format` class.”

The easy and direct access to external information on reusable components provided by CodeBroker caused many changes of original implementation plans described by the subjects before each experiment. This is best illustrated by the example we used in the scenario. The subject (S5), who did not even know the existence of the

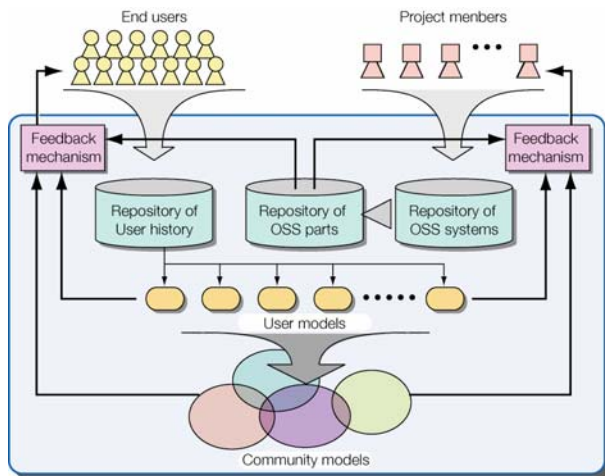


Figure 4: Architecture of the UmCm Engine

`java.text` package, planned “to parse the number, take out the commas and insert the commas.” As he started programming, however, he changed his original plan after he noticed the delivered component `format` from the `java.text.NumberFormat` class. As a result, he created a program similar to that of S3, who anticipated that some methods from the `java.text` package might help him, although he did not know exactly what those methods were, and planned to use them if they were delivered by the CodeBroker system. The combination of S5 and CodeBroker made S5 as skilled as S3 who was originally more knowledgeable than S5 in this specific task.

Most subjects, when asked how well CodeBroker supported their programming, appreciated the usefulness of the system (see col. 7 in Table 1 for their ratings).

At the time of the experiments, the functionality of locating example programs and finding experts was not yet supported. Some components in the library included simple examples to explain their usage. During the experiments, we noticed that whenever subjects found such examples, they always read them for further help. However, only very few components are accompanied by examples. As a response to this issue, we added the functionality of locating examples and experts to provide new layers of information to support locating and using components. Further evaluation of the added support is ongoing. Nonetheless, the preliminary evaluation of the system has shown the benefits brought by the design principle of layered information-on-demand in enabling the active participation of external information resources to complement the insufficient knowledge of programmers.

5. Related Work

The two key concepts in the design principle of *layered information-on-demand* is to deliver task-relevant information and to present information in layers, and do both on demand. The delivery mechanism has

been widely employed in software agents that utilizes users' working context to find task-relevant information, such as design critics [9], remembrance agents [21], and reconnaissance agents [14]. However, such agents have focused only on the locating of relevant information, without carefully considering how the located information should be presented to users based on human attention being the scarce resource.

Structuring information into different layers and presenting each layer on demand to accommodate the different needs of users has its root in the concepts of progressive disclosure and details on demand [23]. *Progressive disclosure* is a design principle to reduce the complexity of user interfaces by presenting the most common choices to users while hiding more complex choices or additional information, which is displayed when needed. *Details-on-demand* is widely applied in information visualization systems that allow users to obtain details about a selected element from the visualized overview of an information set. Systems that support progressive disclosure and details-on-demand do not take the context of information use into consideration. In contrast, layered information-on-demand is tightly integrated with the workspace of users. Furthermore, it expands the notion of information layers by including examples and peer support that are critical for information use, rather than providing information only.

6. Discussion

To illustrate the value of layered information-on-demand, we discuss its possible application to two widely used systems: web browsers and Microsoft Word.

6.1 Web Browsers and Layered Information-on-Demand

Current web browsers have superficial support for *layered information-on-demand*. The links in an HTML page are the first layer of information that is presented to users when the page is loaded into a browser. The second layer is the URL address that is shown when users place the pointer over a link. However, the URL address does not provide enough information about the contents of the linked page. With its current interface, users have to click the link and wait for the browser to take them to the new page, often finding it uninteresting or a dead link.

Applying the principle of *layered information-on-demand* systematically to web browsers can create a smoother interaction to reduce user frustration. While users are reading the current page, the browser could use the time to pre-fetch some portion of the linked pages. When users move the pointer over the link, such pre-fetched results could be shown with a floating label, giving the user a preview of the page, or informing the user that the link does not exist. Previewing a linked page can reduce the cost of jumping back and forth among

links, and eliminate the frustration of visiting dead links. More importantly, users can evaluate the potential value of the link within the context from which it is linked. Especially when several potentially interesting links exist, it makes it easier for users to compare them by simply moving the pointer, and choose the most interesting one.

6.2 Microsoft Word and Layered Information-on-Demand

Certain aspects of *layered information-on-demand* are already supported by Microsoft Word. When a user places the pointer over a menu button, the system shows its text label that is more informative than the button. We believe that a more systematic support of layered information-on-demand described as follows could make it easier for users to learn new functionality.

User puts the pointer on a button.

A text label is shown.

Based on the detection of hesitation (such as the pointer remains on the button or moves toward the label), a more detailed explanation, such as the one that is shown by the *Shift-F1* key (the "What's This" functionality) should be displayed.

A direct transition from this *Shift-F1* explanation to the full help system, which provides more systematic explanation and links to other concepts and functionality, should be provided.

When the user needs further support, an expert list on this functionality should be found by using a mechanism similar to the OWL system [19], which knows experts on a command by tracking their use of the system.

7. Summary

In this paper, we discussed the design principle of *layered information-on-demand* that guides the location and presentation of information by taking into consideration users' task at hand, individual difference of information needs, and information-mediated peer support. We presented the design, implementation, and evaluation of the CodeBroker system to show how the implementation of the principle supports Java programmers by providing smooth transition from programming to acquiring external information and knowledge to complement their insufficient knowledge. Our future work includes further evaluation of CodeBroker and applying the principle to different domains as we have sketched in the previous discussion section.

Acknowledgements. The research was supported by NSF Grants #REC-0106976 and #CCR-0204277; SRA Key Technology Laboratory, Inc., Tokyo; the Coleman Initiative, San Jose, CA; Japan Science and Technology Corporation, Research and Development for Applying

References

1. Belkin, N.J. Helping People Find What They Don't Know. *CACM*, 43 (8), 2000, 58-61.
2. Berlin, L.M. Beyond Program Understanding: A Look at Programming Expertise in Industry. in Cook, C.R., Scholtz, J.C. and Spohrer, J.C. eds. *Empirical Studies of Programmers: Fifth Workshop*, Ablex Publishing Corporation, Palo Alto, CA, 1993, 6-25.
3. Buxton, W. Less is More (More or Less). in Denning, P.J. ed. *The Invisible Future: The Seamless Integration of Technology in Everyday Life*, McGraw-Hill, New York, 2001, 145-179.
4. Carberry, S. Techniques for Plan Recognition. *User Modeling and User-Adapted Interaction*, 11, 2001, 31-48.
5. Carey, T. and Rusli, M. Usage Representations for Reuse of Design Insights: A Case Study of Access to On-Line Books. in Carroll, J.M. ed. *Scenario-Based Design: Envisioning Work and Technology in System Development*, Wiley, New York, 1995, 165-182.
6. Dieberger, A., Dourish, P., Hink, K., Resnick, P. and Wexelblat, A. Social Navigation: Techniques for Building More Usable Systems. *Interactions*, 7 (6), 2000, 36-45.
7. Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S. and Harshman, R., Using Latent Semantic Analysis to Improve Access to Textual Information. in *Human Factors in Computing Systems*, (Washington, D.C., 1988), ACM, 281-285.
8. Fischer, G., Supporting Learning on Demand with Design Environments. in *International Conference on the Learning Sciences*, (Evanston, IL, 1991), Association for the Advancement of Computing in Education, 165-172.
9. Fischer, G., Nakakoji, K., Ostwald, J., Stahl, G. and Sumner, T. Embedding Critics in Design Environments. in Maybury, M.T. and Wahlster, W. eds. *Readings in Intelligent User Interfaces*, Morgan Kaufman, San Francisco, CA, 1998, 537-561.
10. Fischer, G. and Ye, Y., Personalizing Delivered Information in a Software Reuse Environment. in *Proceedings of 8th International Conference on User Modeling*, (Sonthofen, Germany, 2001), Springer-Verlag, 178-187.
11. Galegher, P., Kraut, R. and Egido, C. (eds.). *Intellectual Teamwork*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1990.
12. Hollan, J., Hutchins, E. and Kirsch, D. Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research. in Carroll, J.M. ed. *Human-Computer Interaction in the New Millennium*, ACM Press, New York, 2001, 75-94.
13. Lieberman, H., Fry, C. and Weitzman, L. Exploring the Web with Reconnaissance Agents. *CACM*, 44 (8), 2001, 69-75.
14. Linton, F., Charron, A. and Joy, D. OWL: A Recommender System for Organization-Wide Learning. *Educational Technology & Society*, 3 (1), 2000.
15. Nardi, B.A., Miller, J.R. and Wright, D.J. Collaborative, Programmable Intelligent Agents. *CACM*, 41 (3), 1998, 96-104.
16. Rhodes, B.J. and Maes, P. Just-in-time Information Retrieval Agents. *IBM Systems Journal*, 39 (3&4), 2000, 685-704.
17. Salomon, G. (ed.), *Distributed Cognitions: Psychological and Educational Considerations*. Cambridge University Press, Cambridge, United Kingdom, 1993.
18. Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer-Interaction*, 3rd edition. Addison-Wesley, Reading, MA, 1998.
19. Simon, H.A. *The Sciences of the Artificial*, Third edition. The MIT Press, Cambridge, MA, 1996.
20. Soloway, E. and Ehrlich, K. Empirical Studies of Programming Knowledge. *IEEE ToSE*, SE-10 (5), 1984, 595-609.
21. Terveen, L.G. An Overview of Human-Computer Collaboration. *Knowledge-Based Systems*, 8 (2-3), 1995, 67-81.
22. Ye, Y. Supporting Component-Based Software Development with Active Component Repository Systems. Ph.D. Dissertation, Department of Computer Science, University of Colorado, Boulder, CO, 2001.
23. Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., Kusumoto, S., Component Rank: Relative Significance Rank for Software Component Search, *Proceedings of the 25th International Conference on Software Engineering (ICSE2003)*, pp14-24, Portland, Oregon, U.S.A., May 6-8, 2003.
24. Mili, A., R. Mili, et al. (1998). A Survey of Software Reuse Libraries. In *Systematic Software Reuse*. W. Frakes, (Ed.) Bussum, The Netherlands: Baltzer Science, pp. 317-347.
25. Nakakoji, K., Fischer, G., Intertwining Knowledge Delivery, Construction, and Elicitation: A Process Model for Human-Computer Collaboration in Design,"*Knowledge-Based Systems Journal: Special Issue on Human-Computer Collaboration*, Vol.8, No.2-3, Butterworth-Heinemann Ltd, Oxford, England, pp. 94-104, 1995.
26. Nakakoji, K., Yamada, K., Yamamoto, Y., Morita, M., A Conceptual Framework for Learning Experience Design, *Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C5 2003)*, Kyoto, Japan, pp.76-83, January, 2003.

27. Poulin, J. S. (1999). Reuse: Been There, Done That. *Communications of the ACM*, 42(5): 98-100.
28. Ye, Y., G. Fischer, et al. Integrating Active Information Delivery and Reuse Repository Systems. *Proceedings of ACM SIGSOFT 8th International Symposium on Foundations of Software Engineering (FSE8)*, San Diego, CA, pp. 60-68, 2000.
29. Ye, Y., Fischer, G., Supporting Reuse by Delivering Task-Relevant and Personalized Information, *Proceedings of 2002 International Conference on Software Engineering (ICSE'02)*, Orlando, FL, pp. 513-523, May 19-25, 2002.