CodeBroker: An Active Reuse Repository System

Yunwen Ye

Mar 15, 2004

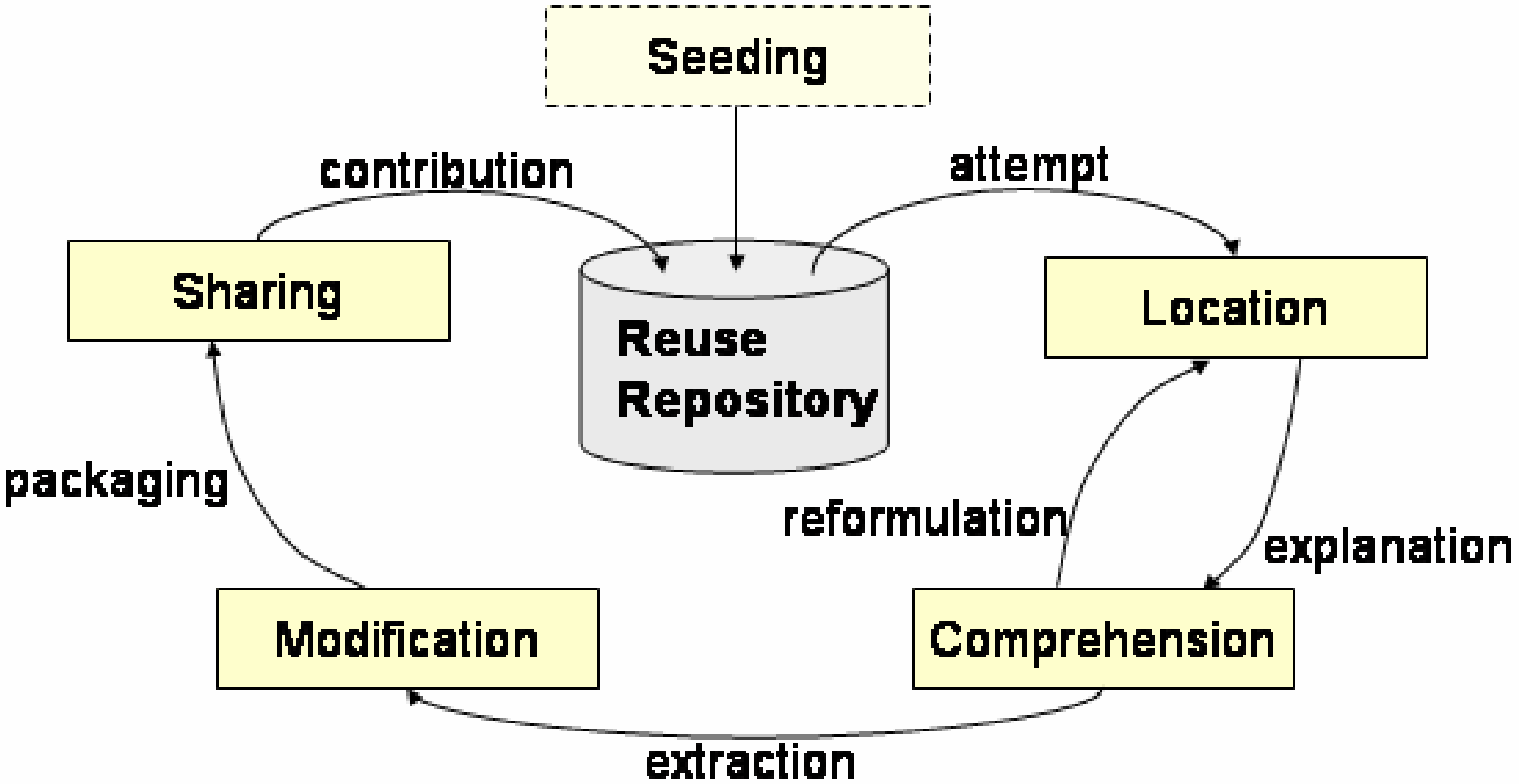
Software reuse

- Definition
  - Creating new software systems with existing artifacts
- Reusable artifacts
  - Code artifacts
    - macros, functions, ***methods, classes***, subsystems, systems
  - Non-code artifacts
    - analyses, designs, test plans and cases, domain models
  - Knowledge
    - program idioms, program plans, design patterns, software architecture styles, domain knowledge
- Reuse repository systems
  - Supporting reuse activities

Why reuse?

- Increased productivity
  - Reduced development time
  - Reduced cognitive load
  - Reduced testing time
- Increased quality
  - Fewer bugs
- Enhanced evolvability and maintainability

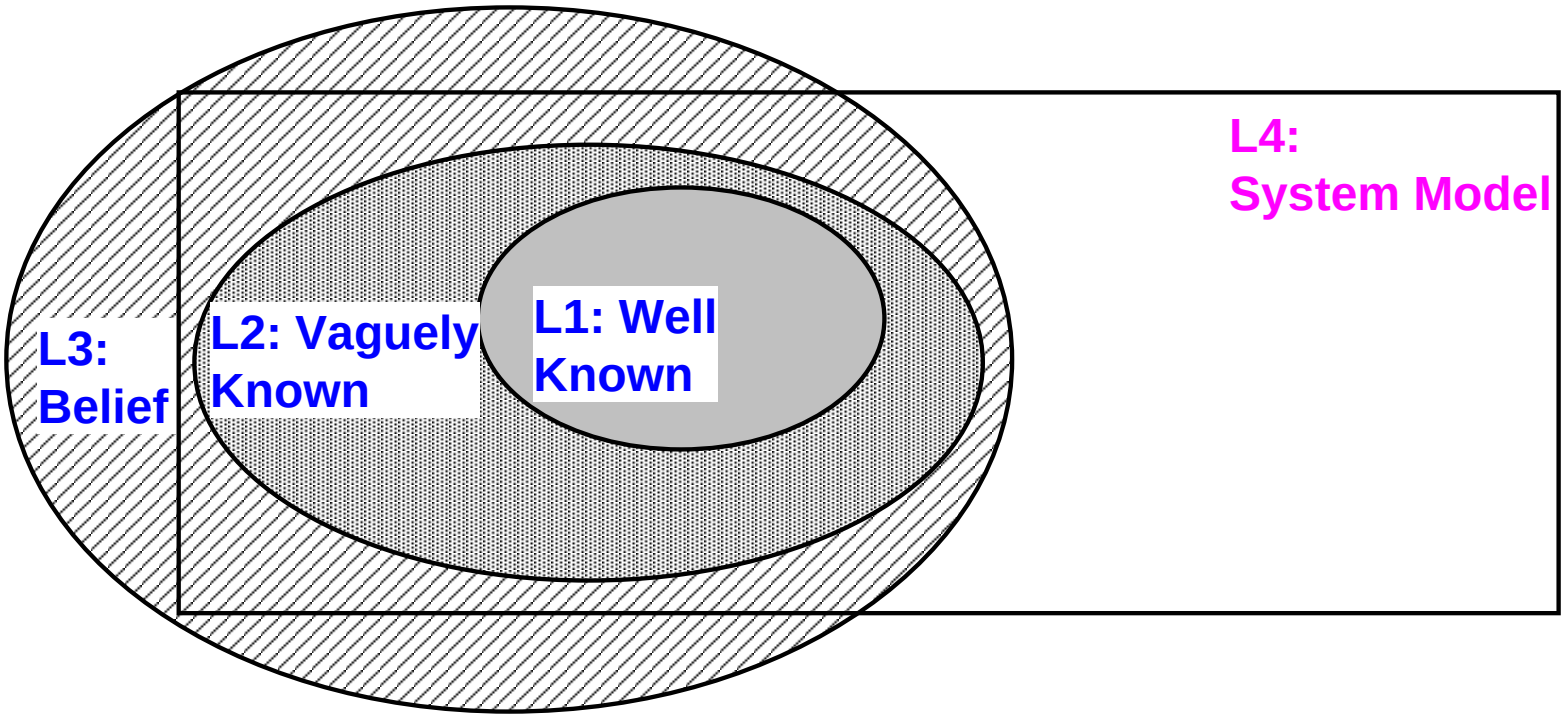
Reuse process (sLCMS)



Research problems

- No attempt to reuse (*Location*)
  - Information islands
    - Not aware of the existence of reusable components
  - Perceived low reuse utility (benefits/cost)
    - High cost of locating components
- Unable to locate the component (*Location*)
  - Situation model vs. system model
- Unable to use the component (*Comprehension*)

User's knowledge about a reuse repository



Reinventing the wheel

- **Have you ever found that you have accidentally implemented a function that is in the library already?**
- Countless times! (tomo)
- Yes this happens often while learning a new language. (prabhu)
- Yes, I have done this a number of times. (mandalia)
- Yup, I wrote a parser in Java that would have been much easier with a StringTokenizer. I'm sure I've done this other times, but that one really gets me (minick).
- Yes. When I was trying to convert a string of numbers into integer, I wrote a function to do it. Later on I found out there is function atio in C library to the exactly the same thing (jing).
- Probably many times, but how would I know? (Jon Marbach).

Reinventing the wheel

- **Have you ever found that you have accidentally implemented a function that is in the library already?**
- Not yet (jackson)
- I cannot remember ever implementing a function that was already in the library. (deriggi)
- No, but I have never really checked this out (Serina Croll).

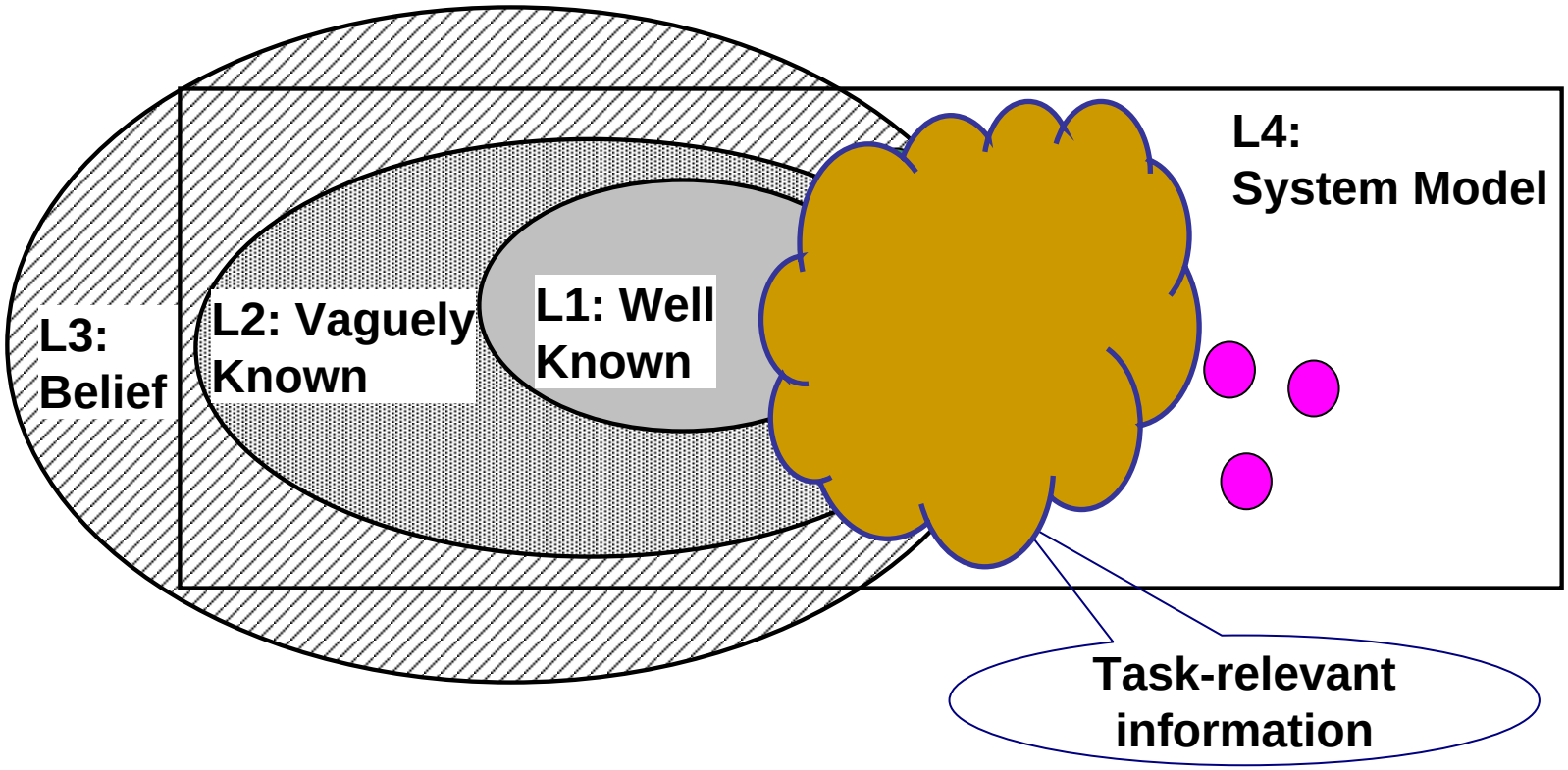
Reinventing the wheel

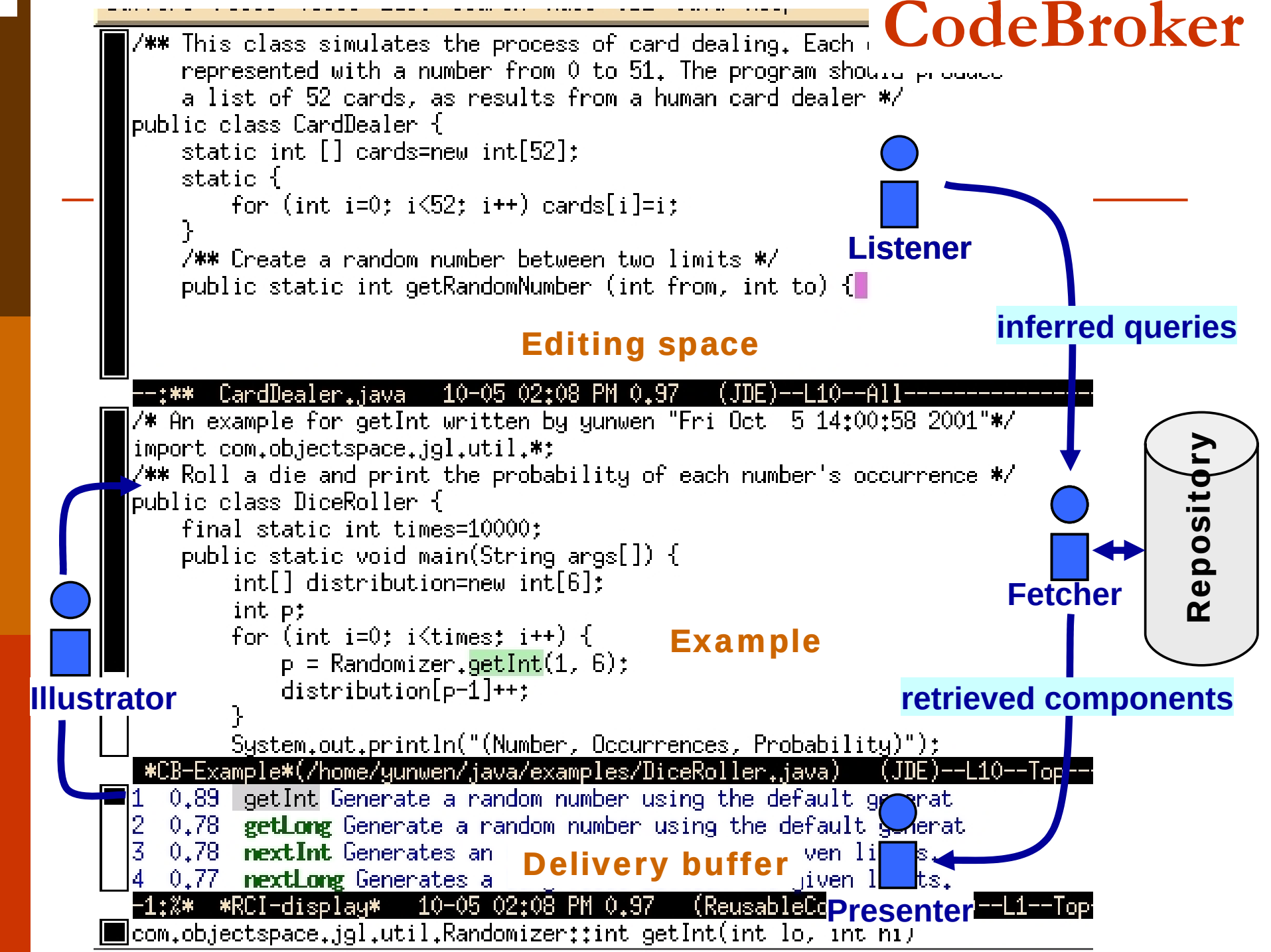
- “Conversations with developers revealed several cases in which programmers, unaware of a virtual machine primitive for an operation, repeatedly reimplemented the same operation--in one case, *ten times*.” [Devanbu, 1991]
- Reusable objects demand *proper advertisement* [Walton, 1992]
- “We have discovered that ‘*marketing’ the components* in the CSL is just as important as providing the correct technologies for users in Schlumbeger Oilfield Services products. [Rosenbaum 1995]
- It happens that we develop functions when they exist and we do not realise it. [Coulange 1997]
- “I could be creating a method that does exactly the same thing somebody else’s does … even though we have access to each other’s code. We might call them different names and we might have a bit different way of doing it, but we’re still doing the same thing.” [Fichman, Kemerer, 1997]

Proposed solution

- Active component repository systems
  - Overcoming the limits of browsing and searching
  - Supporting information delivery
- Benefits
  - Reusing unknown components
  - Reduced locating cost
  - Seamless integration with programming environment

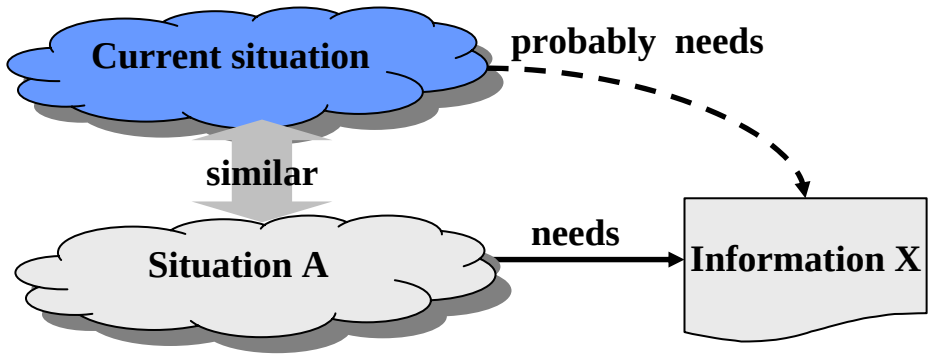
Challenges in active reuse repository systems



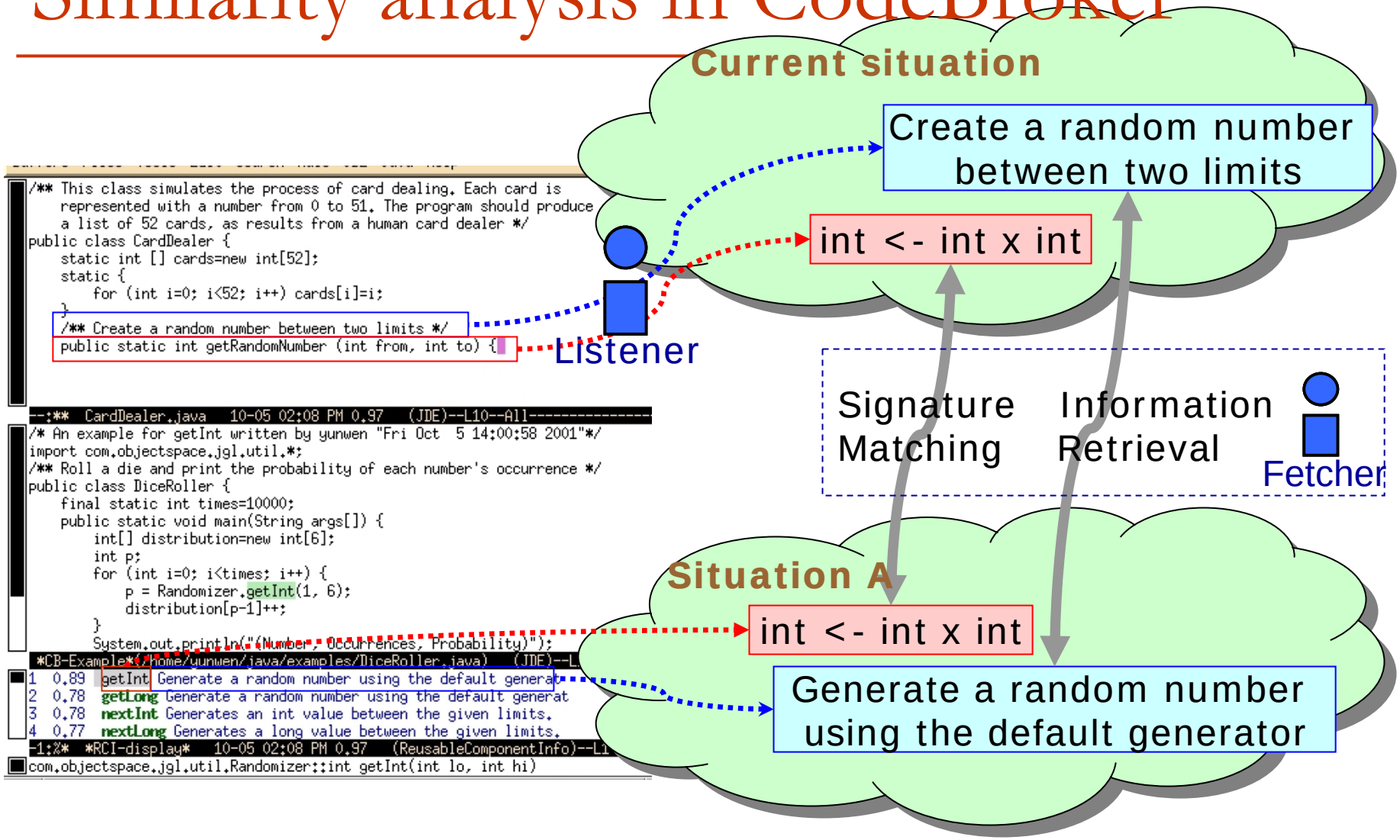

CodeBroker
/** This class simulates the process of card dealing. Each
represented with a number from 0 to 51. The program
a list of 52 cards, as results from a human card dealer */
public class CardDealer {
static int [] cards=new int[52];
static {
for (int i=0; i<52; i++) cards[i]=i;
}
/** Create a random number between two limits */
public static int getRandomNumber (int from, int to) {
Listener
inferred queries
Editing space
--:** CardDealer.java 10-05 02:08 PM 0.97 (JDE)--L10--All
/* An example for getInt written by yunwen "Fri Oct 5 14:00:58 2001"*/
import com.objectspace.jgl.util.*;
/** Roll a die and print the probability of each number's occurrence */
public class DiceRoller {
final static int times=10000;
public static void main(String args[]) {
int[] distribution=new int[6];
int p;
for (int i=0; i<times; i++) {
p = Randomizer.getInt(1, 6);
distribution[p-1]++;
}
System.out.println("(Number, Occurrences, Probability)");
Fetcher
Repository
Example
Illustrator
retrieved components
CB-Example(/home/yunwen/java/examples/DiceRoller.java) (JDE)--L10--Top
1 0.89 getInt Generate a random number using the default generat
2 0.78 getLong Generate a random number using the default generat
3 0.78 nextInt Generates an
4 0.77 nextLong Generates a
Delivery buffer
Presenter
-1:%* *RCI-display* 10-05 02:08 PM 0.97 (ReusableCo --L1--Top
com.objectspace.jgl.util.Randomizer::int getInt(int lo, int hi)

Inferring the task

- Plan recognition
  - Actions → Inferred goal → Suggested actions or information
- Similarity analysis



Similarity analysis in CodeBroker



The rationale

- Three aspects of a program
  - Concept
    - The functionality of the program
    - Semantic information
    - Revealed in *comments*, identifiers, …
  - Constraint
    - Execution environment
    - Syntactic information
    - Revealed in *signatures*, protocols, …
  - Code
    - The implementation
- The assumption
  - Similar concept + compatible signature → reusable code

Basic information retrieval (IR) techniques

- Information retrieval: Finding similar documents based on the commonality of terms
  - Documents and queries are represented by term vectors

    $D_j = (f_{1,j}, f_{2,j}, \ldots, f_{N,j})$
  - Similarity is the distance between two vectors

$$Similarity\ (Q, D) = \sum_{i=1}^{n} Q[i] \times D[i] \Big/ \sqrt{\sum_{i=1}^{n} Q[i]^2 \times \sum_{i=1}^{n} D[i]^2}$$

Term space: (factor information help human operation retrieval system)

	Contents	Vector	Similarity
Q	human factors in information retrieval system	(1 1 0 1 0 1 1)	
D1	factor factor factor human human retrieval system	(3 0 0 2 0 1 1)	$7/75^{0.5}=0.80$
D2	information operation retrieval retrieval	(0 1 0 0 1 2 0)	0.55
D3	factor help help retrieval	(1 0 2 0 0 1 0)	0.37

LSA: Improved IR

- Latent semantic analysis
  - Addressing the vocabulary mismatch problem (people use different names to refer to the same concept)
  - Creating a semantic space with a large amount of documents

$$\begin{pmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,M} \\ w_{2,1} & w_{2,2} & \dots & w_{2,M} \\ \dots & \dots & \dots & \dots \\ w_{N,1} & w_{N,2} & \dots & w_{N,M} \end{pmatrix} = \begin{pmatrix} t_{1,1}^{(0)} & t_{1,2}^{(0)} & \dots & t_{1,r}^{(0)} \\ t_{2,1}^{(0)} & t_{2,2}^{(0)} & \dots & t_{2,r}^{(0)} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ t_{N,1}^{(0)} & t_{N,2}^{(0)} & \dots & t_{N,r}^{(0)} \end{pmatrix} \times \begin{pmatrix} s_{1,1} & 0 & \dots & 0 \\ 0 & s_{2,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & s_{r,r} \end{pmatrix} \times \begin{pmatrix} d_{1,1}^{(0)} & d_{1,2}^{(0)} & \dots & d_{1,M}^{(0)} \\ d_{2,1}^{(0)} & d_{2,2}^{(0)} & \dots & d_{2,M}^{(0)} \\ \dots & \dots & \dots & \dots \\ d_{r,1}^{(0)} & d_{r,2}^{(0)} & \dots & d_{r,M}^{(0)} \end{pmatrix}$$

Reducing the singular vectors

$$\hat{X} = \begin{pmatrix} t_{1,1} & t_{1,2} & \dots & t_{1,k} \\ t_{2,1} & t_{2,2} & \dots & t_{2,k} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ t_{N,1} & t_{N,2} & \dots & t_{N,k} \end{pmatrix} \times \begin{pmatrix} s_{1,1} & 0 & \dots & 0 \\ 0 & s_{2,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & s_{k,k} \end{pmatrix} \times \begin{pmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,M} \\ d_{2,1} & d_{2,2} & \dots & d_{2,M} \\ \dots & \dots & \dots & \dots \\ d_{k,1} & d_{k,2} & \dots & d_{k,M} \end{pmatrix}$$

Probabilistic IR model

- Adding weights to each term

  $D_j = (t_{1,j}, t_{2,j}, \ldots, t_{N,j})$

  $t_{i,j} = TRW_i * f_{i,j}$

- Term Relevance Weight

  $TRW_i = \log (p_i \times (1-q_i) / q_i \times (1-p_i))$

  p_i **Probability of the term appearing in relevant documents**

  q_i **Probability of the term appearing in irrelevant documents**

Weighting schema in CodeBroker

$$sim(Q, D_j) = \sum_{i=1}^{T} (\log \frac{N - n_i + 0.5}{n_i + 0.5}) \frac{(k_1 + 1) tf_{i,j}}{K + tf_{i,j}} \frac{(k_3 + 1) qtf_i}{k_3 + qtf_i}$$

N is the number of components
n_i is the number of components whose documents contain the term *ti*
T is the number of terms in the component collection
$tf_{i,j}$ is the frequency of term *ti* in the document of the component *Dj*
qtf_i is the frequency of term *ti* in the query *Q*

$$K = k_1((1-b) + b \cdot dl_j / avdl$$

k_1,k_3,b are empirically determined parameters depending on the nature of the document collection. In *CodeBroker*, k_1 is set to 1.2, k_3 to 1.0, and *b* to 0.75.
dl_j is the length of document D_j
avdl is the average length of all documents in the collection

Signature matching determines the constraint compatibility

- Reusable components must be compatible in signature
  - Signature is the syntactic interface of a module (method and class)
  - Improving the precision of retrieval
- Method level match
  - Exact match

    ```
    Type1 x Type2 -> Type3
    TypeA x TypeB -> TypeC
    <=> Type1=TypeA  AND Type2=TypeB AND Type3=TypeC
    ```

  - Relaxed match
    - Generalization / Specialization / Reorder

      string x int -> int **matches** (**relaxed**) long x string -> long

Signature matching for classes

```
public class AutomaticReception extends Vector {

public boolean initialize();
public void delete();
public insert(string person);
public int length();
}
```

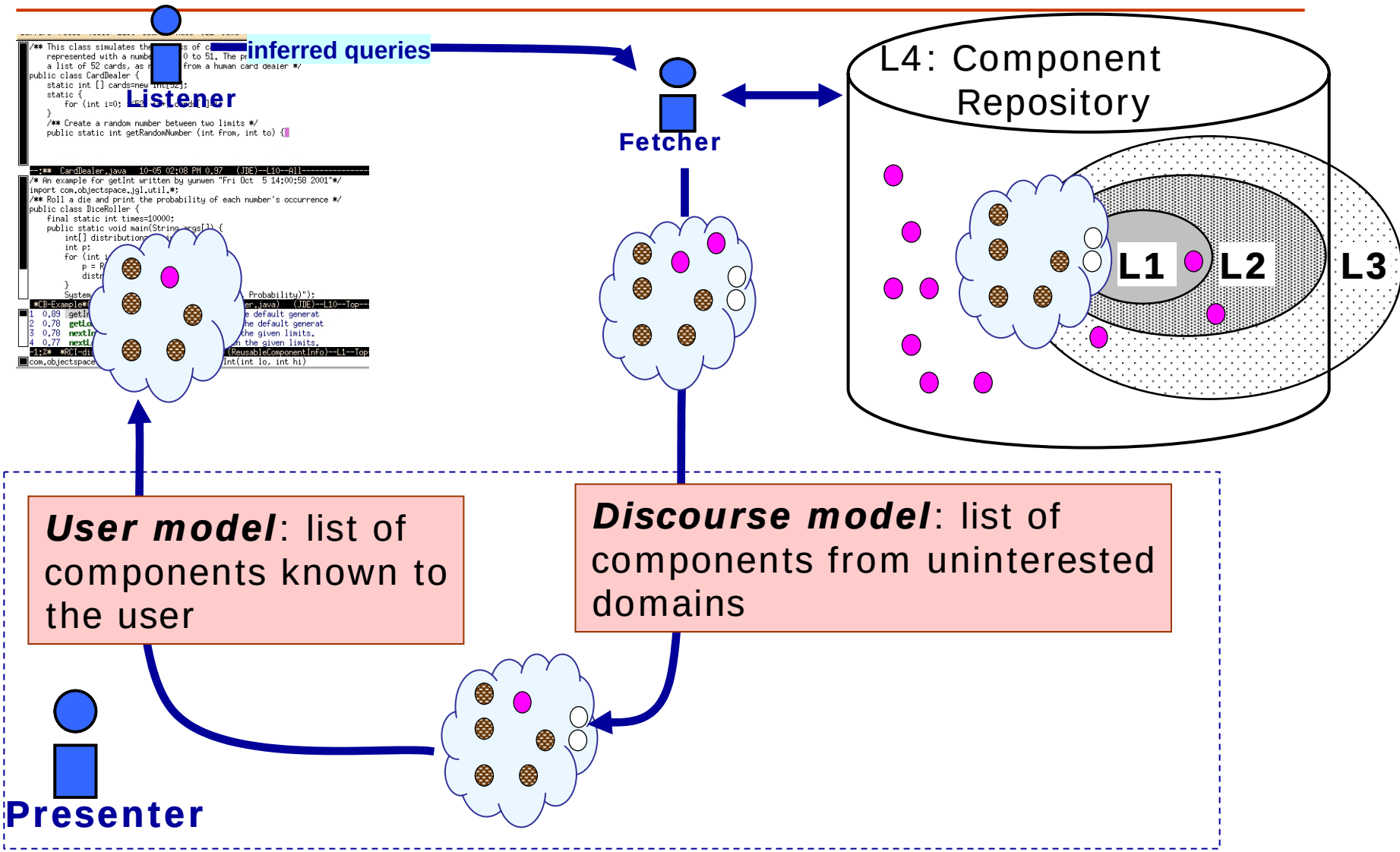
void -> boolean
void -> void
string -> void
void -> int

```
    public class Queue extends Vector
{

    public boolean empty();
    public dequeue();
    public enqueue(Object item);
    public int size();
    }
```

void -> boolean
void -> void
object -> void
void -> int

Presenter: tailoring the delivery to larger context and user



Discourse models: Improving task-relevance

- Discourse models capture the larger context of programming activities
  - Representing the interaction history between programmers and CodeBroker
  - Removing irrelevant components
  - Negative discourse models: specifying what is not of interest to programmers
  - Example:

```
((“java.util.zip”) ;; a package
 (“java.awt” (“CardLayout”))) ;; a class
```

User models: User-specific delivery

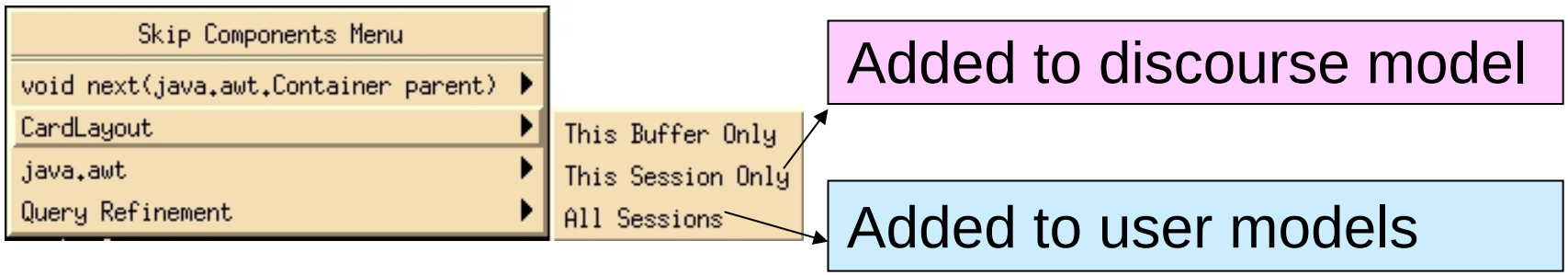
- User models represent programmers' knowledge on the component repository
  - A list of known components
  - Example:

```
(("java.applet" ("Applet" ("getParameterInfo"))
 ("java.io" ("File" ("exists"
                     "11/02/00" "11/10/00"
                      "11/11/00")
                    ("isAbsolute"
                     "11/01/00" "11/10/00"
                     "11/11/00"))))
```

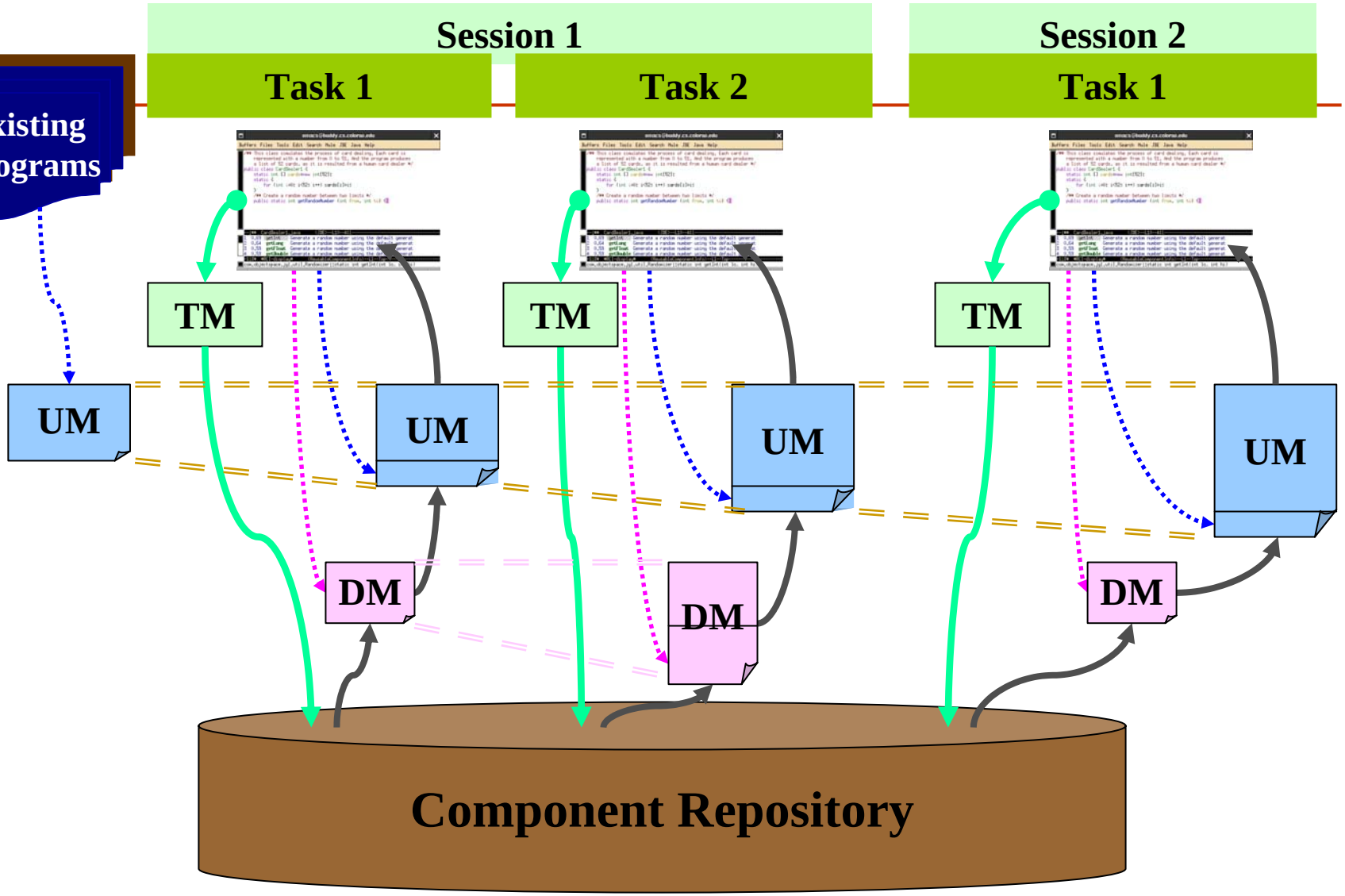
  - Components contained in user models are **not** delivered

Incremental discourse modeling and user modeling

- **Initial** user models
  - Created by analyzing existing user programs
- **Adaptive** user models
  - CodeBroker updates user models automatically when it detects the use of a component in the editor
- **Adaptable** user models and discourse models
  - Using the `Skip Components Menu` associated with each delivered component

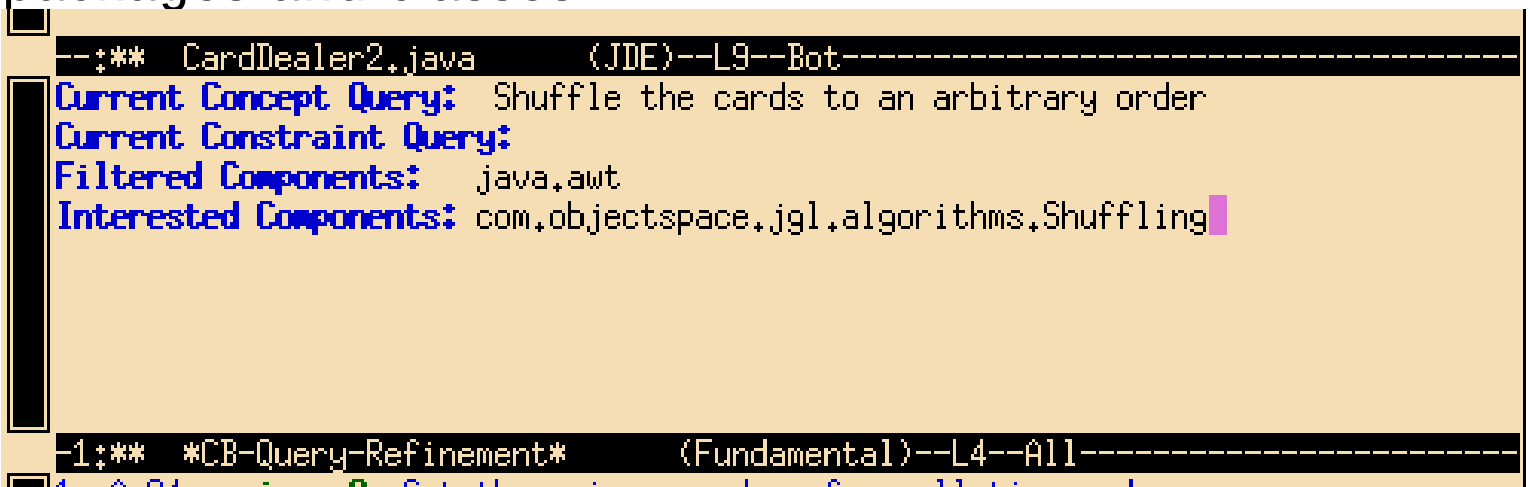


Models in CodeBroker



Retrieval-by-reformulation

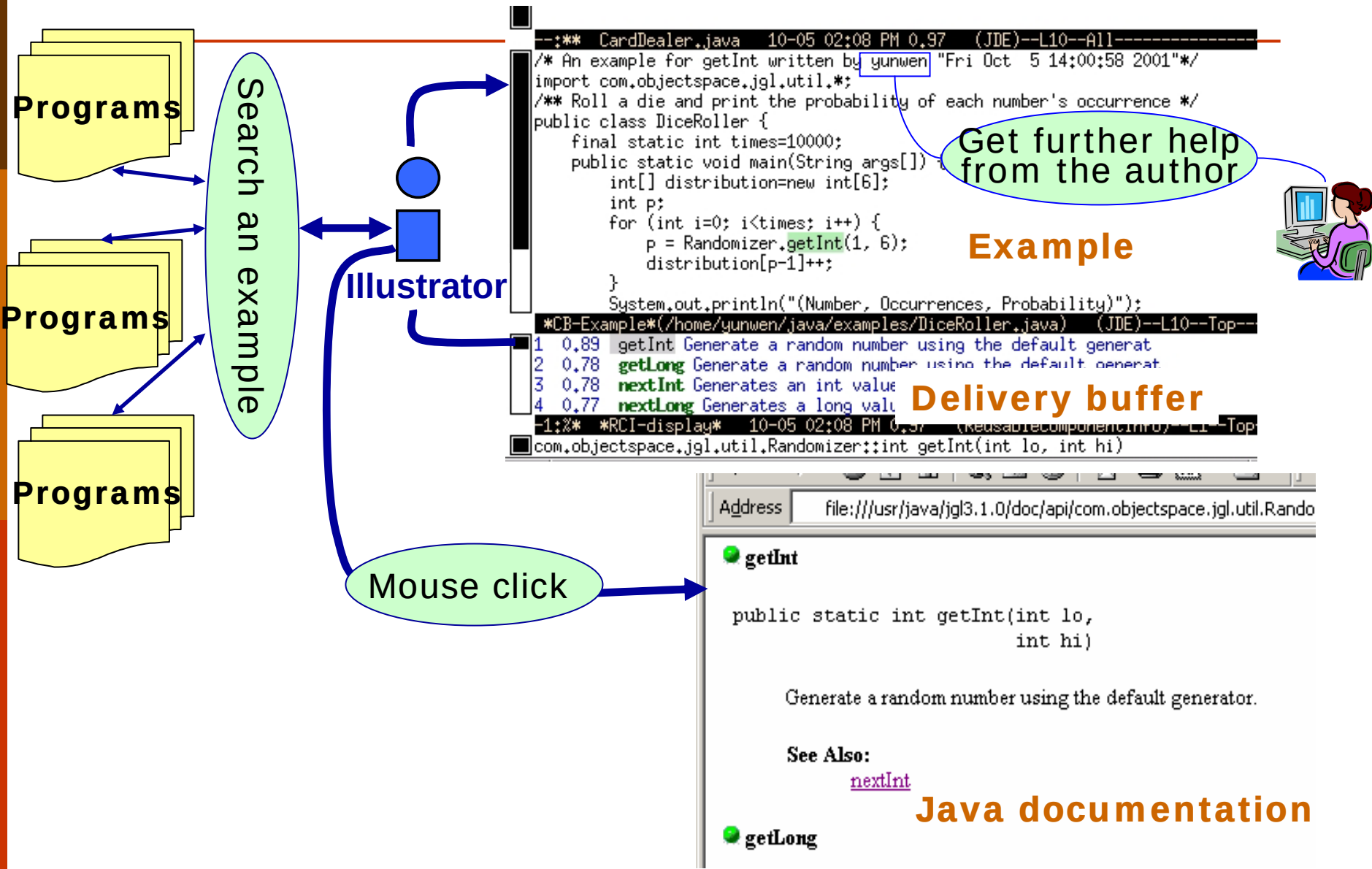
- A process for software developers to incrementally develop reuse queries
- Delivered components help developers become familiar with the vocabulary and structure of the repository
  - Change the way of writing the query
  - Limit the search scope by specifying (un)interested packages and classes



The cycle of delivery-browsing-searching

- Delivered components are results of information reconnaissance
- Possible actions after the delivery
  - The needed component is delivered
    → Choose the needed one through browsing
  - Too many components are delivered
    → Filter the delivered components
  - The needed one is not delivered
    → Search again through retrieval-by-reformulation

Supporting comprehension and use



Evaluating retrieval effectiveness

- Recall =

$$\frac{\text{No. of relevant doc. retrieved}}{\text{No. of relevant doc.}}$$

- Precision =

$$\frac{\text{No. of relevant doc. retrieved}}{\text{No. of doc. retrieved}}$$

- Results of 19 queries
  - One-third is relevant

Recall	Prob. Precision	LSA Precision
0	45.82	35.77
10	45.82	31.86
20	45.82	30.89
30	41.20	25.62
40	41.01	20.62
50	40.74	20.44
60	37.46	13.86
70	37.46	13.82
80	32.71	13.82
90	32.19	12.32
100	29.43	12.32

Evaluation experiments

- Experiment goals:
  - Observe the effectiveness of CodeBroker in encouraging programmers to reuse
  - Analyze the effectiveness of task inference, discourse models, and user models
- 12 experiments with 5 subjects
  - Implementing an assigned task with CodeBroker

Subjects	S1	S2	S3	S4	S5
Years of prog. in general	3-4	5-6	8	10+	10+
Java skill (self-evaluation)	4	7	7-8	10	7

System assessment

Sub	No	total	delivered	breakdown of deliveries			triggered
				unanticipated (L4-L3)	anticipated but unknown (L3)	vaguely known (L2)	
S1	1	10	4	2	2	0	0
	2	3	1	1	0	0	1
S2	3	7	1	1	0	0	0
	4	4	1	1	0	0	0
	5	5	3	0	2	1	1
S3	6	5	2	1	1	0	1
	7	4	3	1	2	0	1
	8	3	0	0	0	0	0
S4	9	4	3	0	3	0	0
	10	3	1	1	0	0	2
S5	11	4	1	1	0	0	2
	12	5	0	0	0	0	0
Sum		**57**	**20**	**9**	**10**	**1**	**8**

Role of discourse models

Subject	Task	Retrieved#	Added to DM#	Removed by DM#
S1	T1	168	1 *pkg.*, 1 *class*	45
	T2	28	1 *pkg.*, 1 *class*	10
S2	T3	140	4 *methods*	0
S4	T3	80	1 *pkg.*	7
	T5	140	2 *pkgs.*	68
Other 7 experiments		872	0	0

- Discourse models removed irrelevant components
- Larger tasks may make programmers add more components to discourse models

Role of user models

Retrieved	Removed	User added	System added
168	15	0	0
28	0	0	0
140	5	0	0
52	0	0	0
160	14	2	5
60	0	0	6
20	1	0	0
60	0	0	0
80	0	0	0
140	0	0	0
100	1	0	9
420	0	0	0

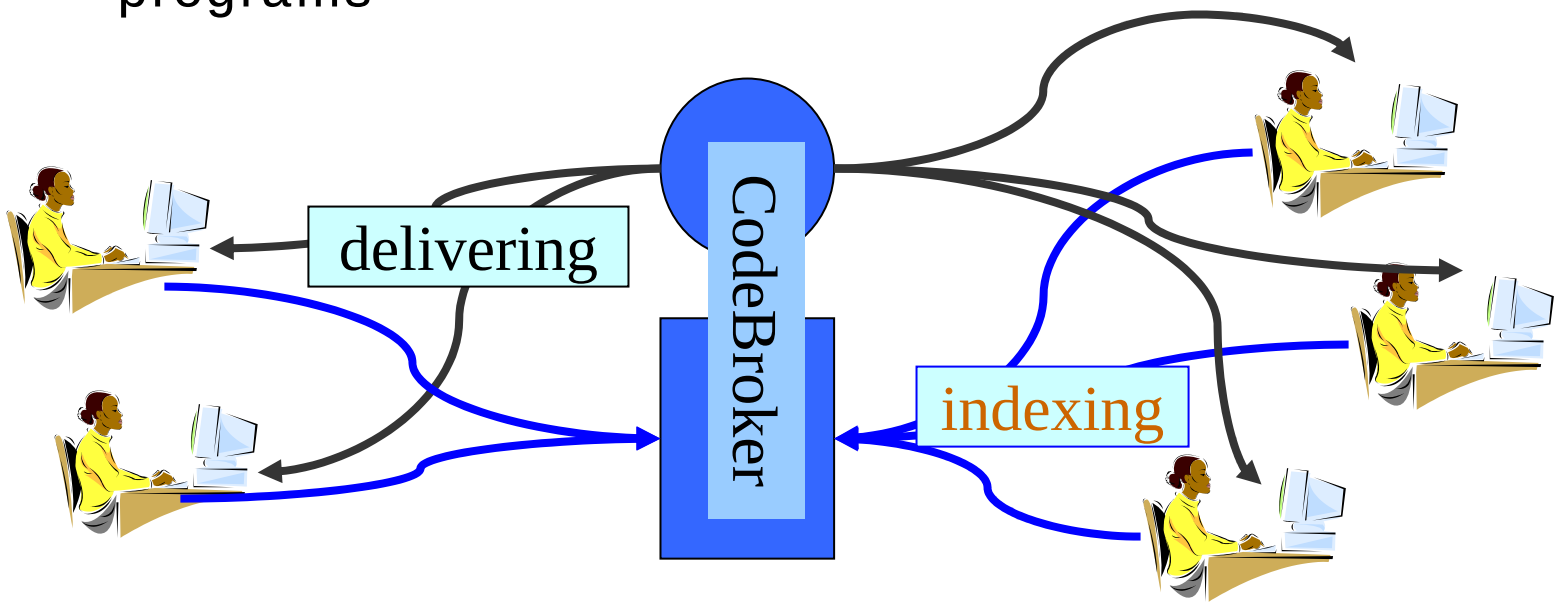
- User models removed few components
  - Incomplete user models
  - Most of the delivered components were unknown
- Removed components not reusable
- User models too simple
  - Unforgiving
  - No decaying mechanism

Problems found

- Irrelevant components
  - → *More sophisticated task modeling techniques*
- Abstraction mismatch from queries to components
  - → *Indexing based on usage*
- Lack of guidance in refining queries
  - → *Guidance on the choice of terms*
- Lack of configurability
  - → *More user-friendly interface*
- Lack of examples
  - → *The development of the Illustrator agent*

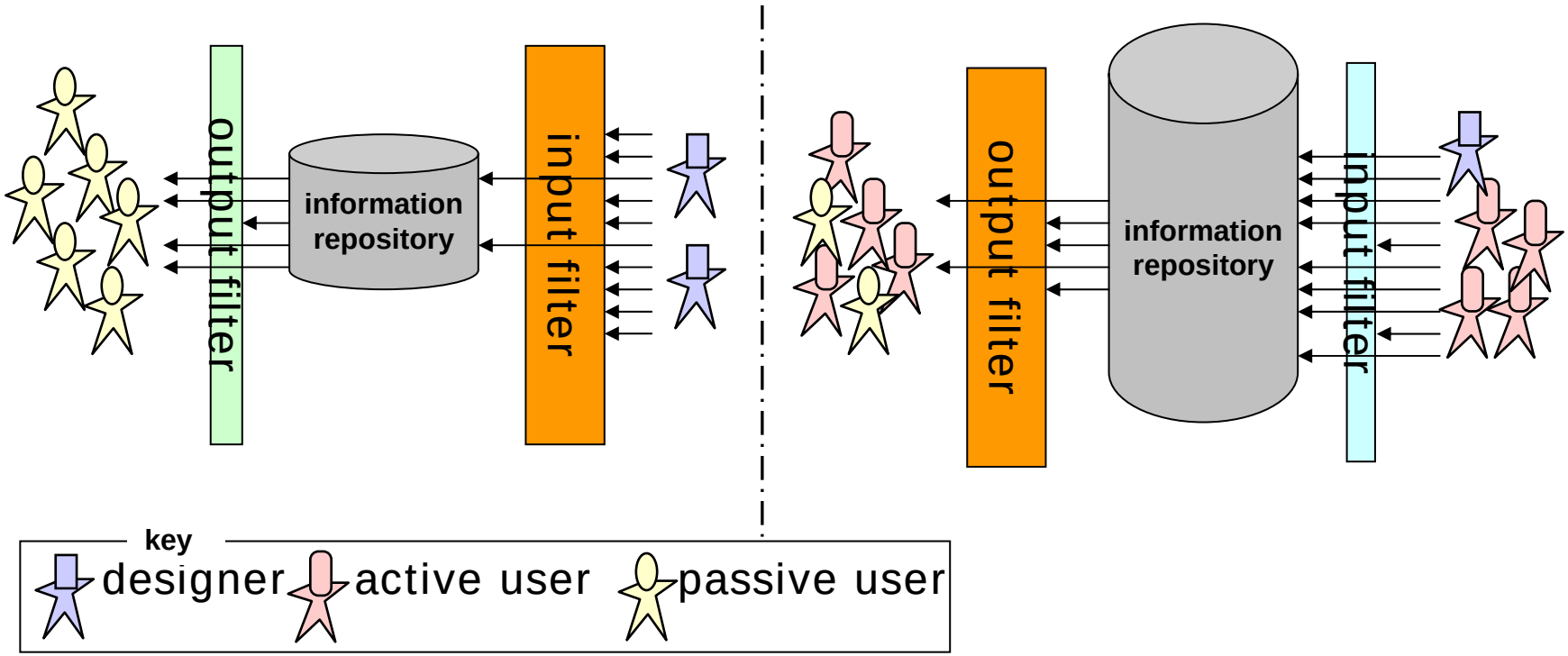
Future research

- **Long-term** user models and their evaluation in natural settings
- **Distributed** CodeBroker supporting software development communities
  - Make programmers aware of reusable components
  - Bring together programmers working on similar programs



General lesson: Designing information repository systems

- Two modes of designing and using information repository
  - Filtering the input vs. Filtering the output



Summary

- Better understanding of cognitive difficulties of component reuse
  - Unknown components
  - Low reuse utility
- A new type of component repository systems
  - Active component repository systems
- Contributions to the design of information repository systems in general
  - Similarity analysis-based task modeling
  - Focusing output filter instead of input filter