安心情報システム構築における コンポーネント技術の応用

Development of Information Systems for e-Society with Component Technologies

鈴木　正人

北陸先端科学技術大学院大学
情報科学研究科

06/11/28 Verite 3rd

Contents

- Requirements of Information systems for e-society (*accountability*)
- Our goal
- Component technologies (Flexibility, Specification&Verification)
- Our approach
- Restructuring current system w/components.
- Current Status/Summary

e-Society

Katayama used the term “Verifiable and Evolvable e-Society” in our COE21 projects.

Features of e-Society

Correctness: all functions must be correctly realized according to its specifications

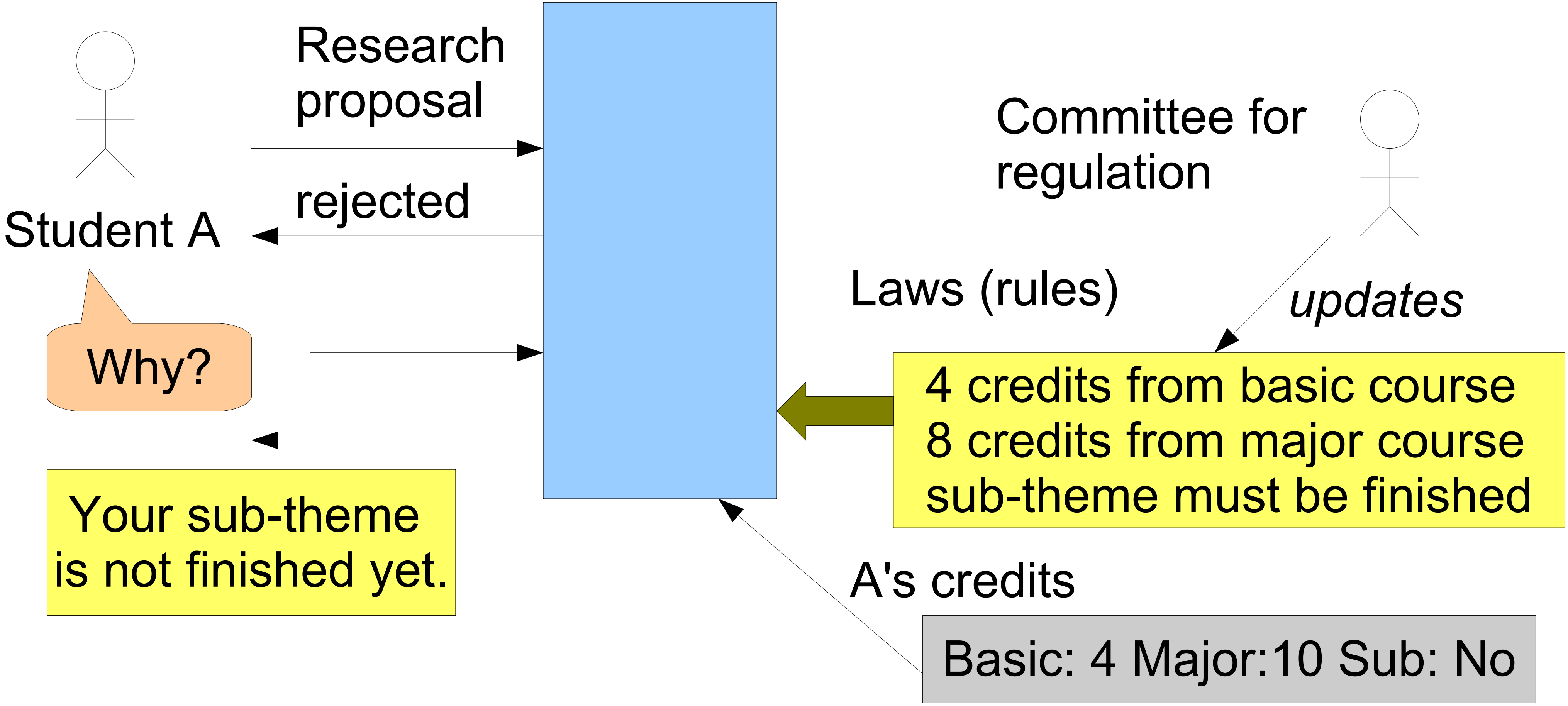
Accountability: systems must explain its functions and structures for all questions by all stakeholders

Security: systems must prohibit leak of information and unauthorized accesses

etc.

Outline of Info. Sys. with accountability

Credit/Score management system in our Institute



Research proposal

rejected

Student A

Why?

Your sub-theme is not finished yet.

Committee for regulation

Laws (rules)

updates

4 credits from basic course
8 credits from major course
sub-theme must be finished

A's credits

Basic: 4 Major:10 Sub: No

Features for Info. Sys. with accountability

System must provide not only the result but a cause or a history of reasoning.

Research proposal

R33: acceptance conditions of research proposal

- R33-1: 4 credits from basic course
- R33-2: 8 credits from major course
- R33-3: sub-theme must be finished

Traditional system only gives answer “rejected”
System with accountability must give answer such that

R33-1: You have 4, requires 4 PASS
R33-2: You have 10, requires 8 PASS
R33-3: You don't finish sub-theme FAILED → Cause of failure
R33 is AND(R33-1, R33-2, R33-3) FAILED
Result : Rejected

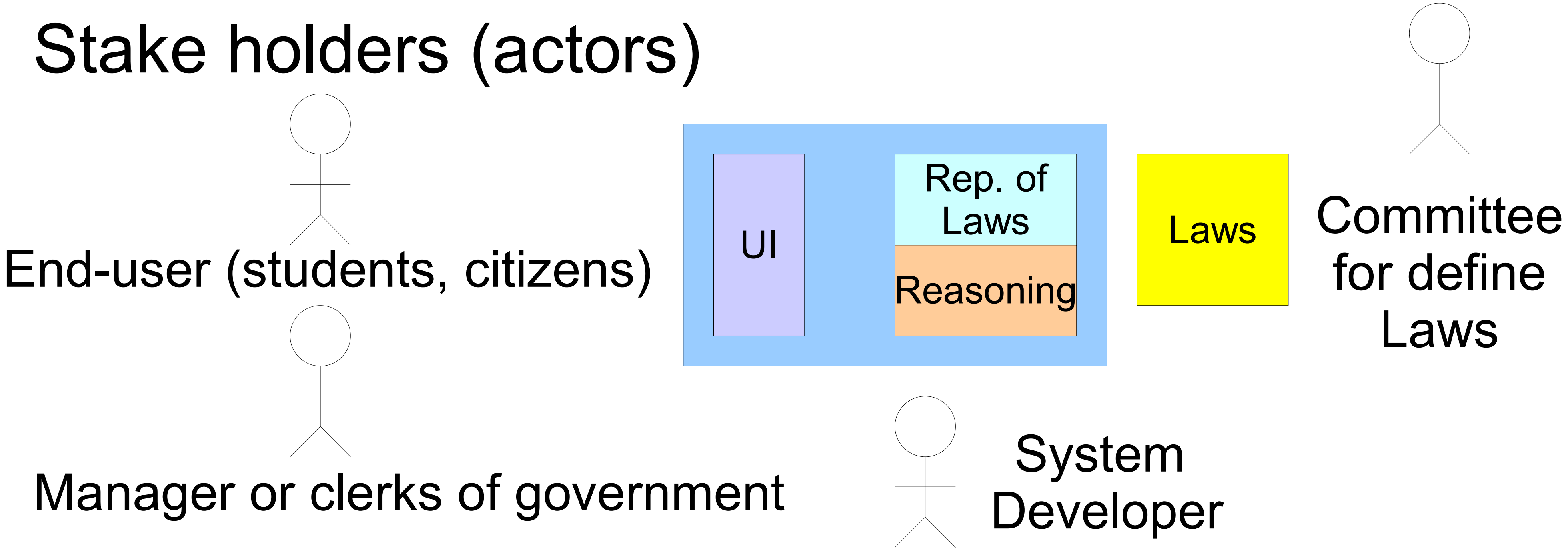
AND-OR tree is used

Our goal

One of our goal is to provide a technical basis for realizing info. sys. with accountability.

(Efficiency in development/evolution, verification, reuse)

Stake holders (actors)



End-user (students, citizens)

UI

Rep. of Laws

Reasoning

Laws

Committee for define Laws

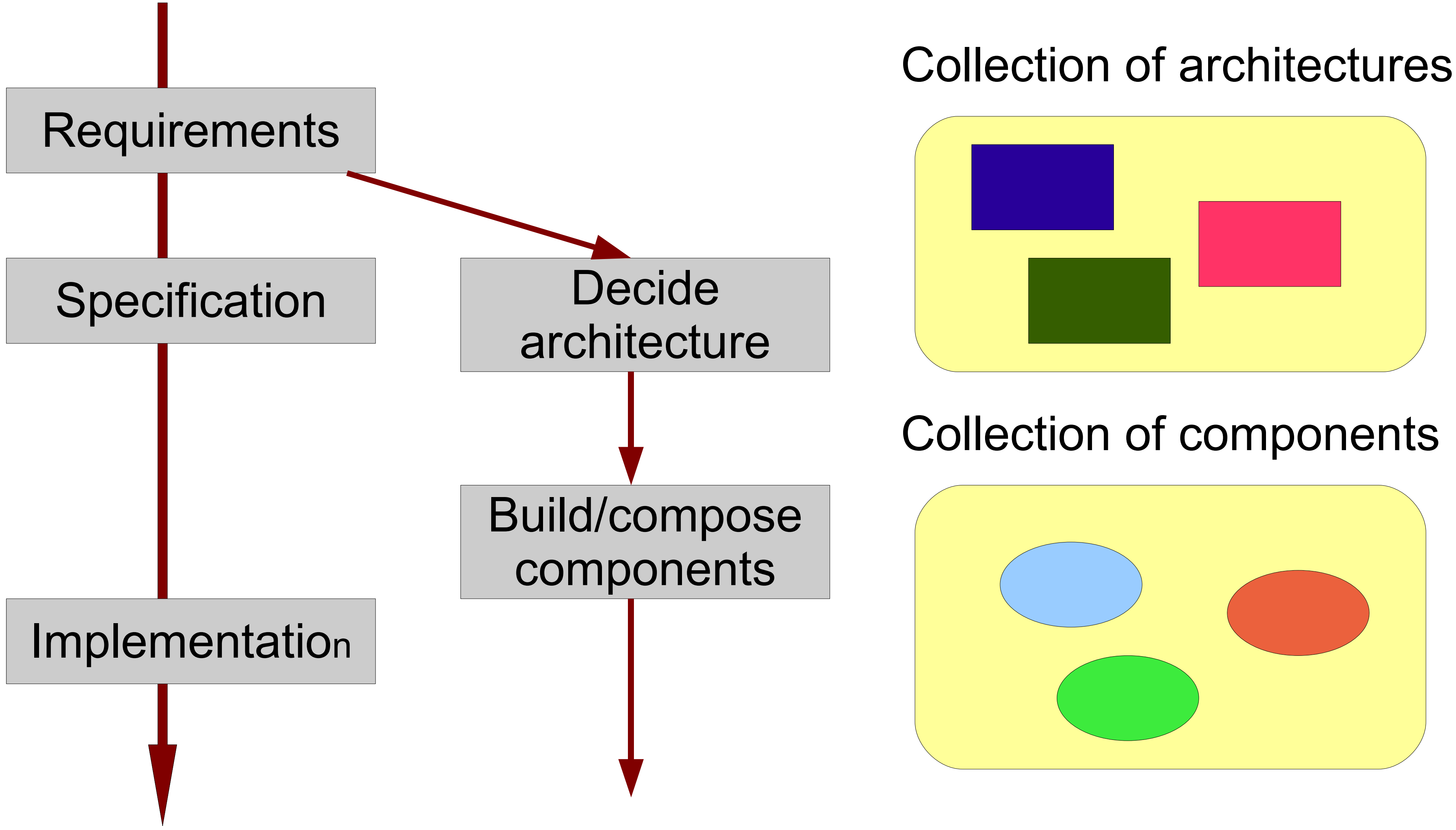
Manager or clerks of government

System Developer

Software architecture/component based technologies may give a proper solution.

Component Technologies

Originally aimed to improve cost/efficiency in reuse.



Requirements

Specification

Implementation

Decide architecture

Build/compose components

Collection of architectures

Collection of components

Features of components

Originally it was any unit of program (modules etc)

Recently it is based on Object-Oriented, and have the following features [Ning 96]

How to use (interfaces) are open to public, but internal structures are hidden.

Works on a particular environment only.

Unit of plug-in (replacement)

Consists of multiple (binary/text) files

Component features for accountability

1. Flexible connection

A component communicate to another one / its environment through some indirect mechanisms.

Traditional:

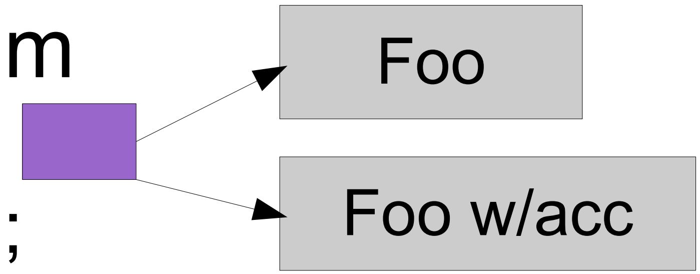
```
Foo( int id, String name)
```

Foo

Caller must know the address of function “Foo.” We have to re-compile all if we change behavior.

With component:

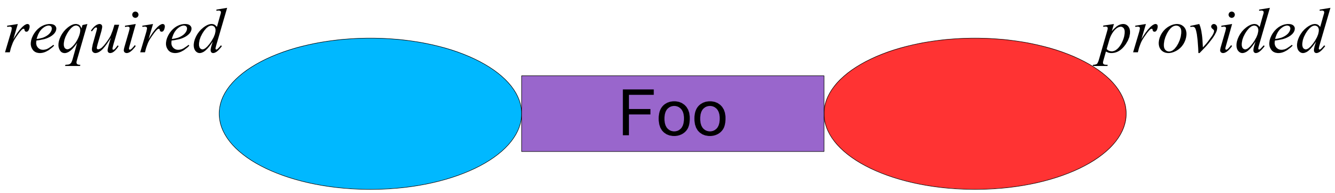
```
Interface i = c.getInterface();
Method m = i.getMethod (“Foo”);
m.invoke(args( id, name )) ;
```



Component features for accountability

2. Specification / Verification

Interfaces and their usage must be verified at compiled time (static) or runtime (dynamic).



required

Foo

provided

Arg1 is int
Arg2 is String

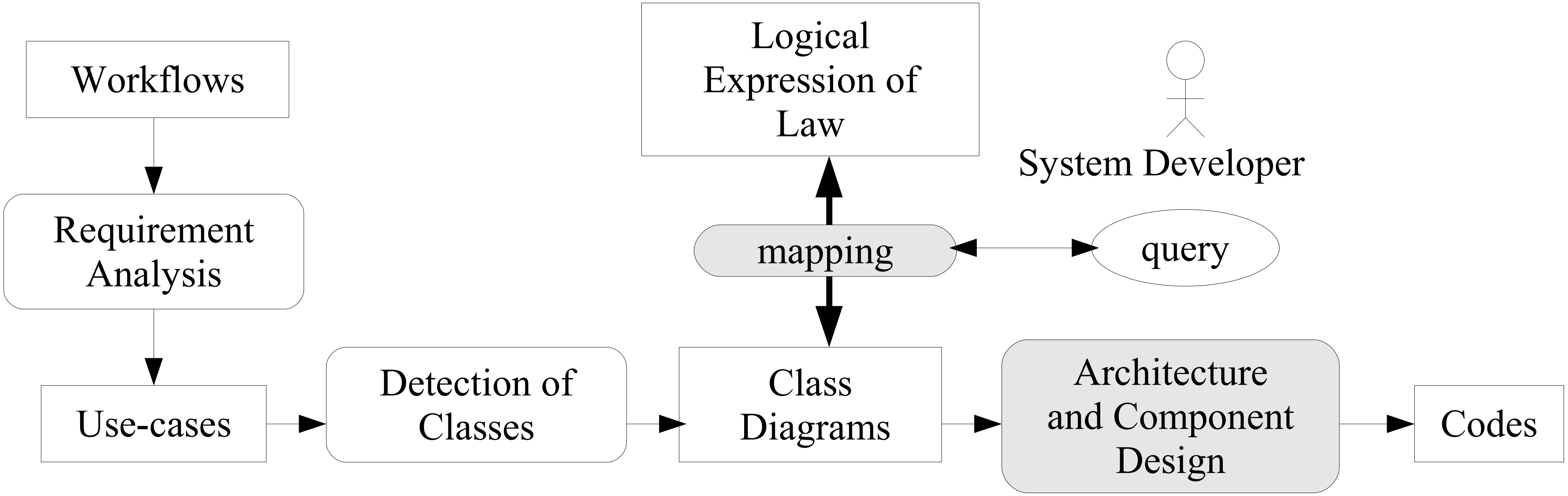
Arg1 must be int, positive
Arg2 must be String

Traditional: spec. must be described separately and independent verifier is required at runtime.

Component technologies already have/easy to extend specification/ semi-automatic verification.

Our approach

Overview of development process



(1) we have to extract candidates of classes from expression of laws.

4 credits from basic course
8 credits from major course
sub-theme must be finished

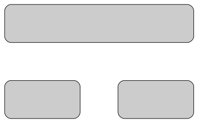
Research proposal

Credit

Sub-theme

Our approach(cont.)

(2) Design classes from use-cases and (1)



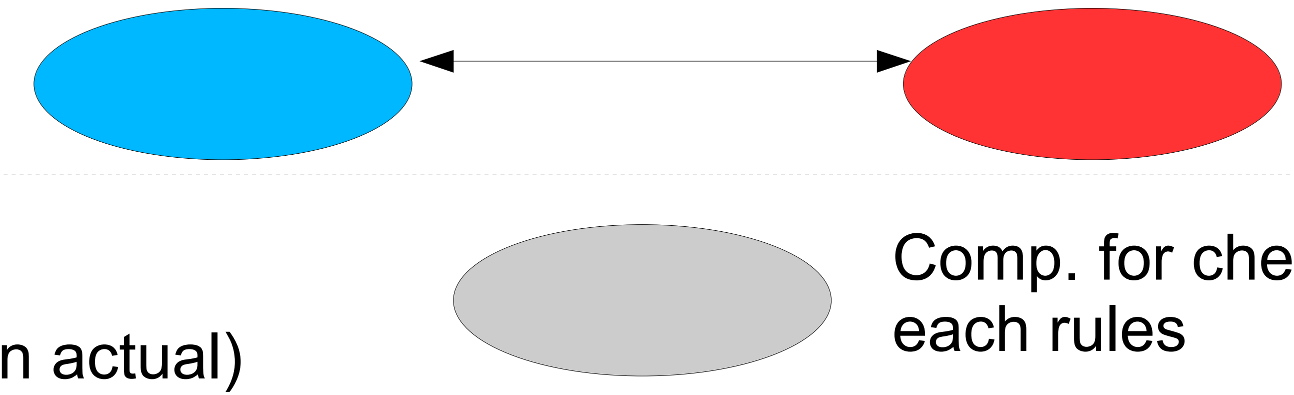
Use-case name:accept research proposal
actors: student, manager
normal sequences:
  1: student gives proposal
  2: system checks conditions by reasoning
  ...

Condition

Query history

Reasoning history

(3) Implement using component models



Comp. for query manager

Comp. for reasoning manager

Comp. for checking each rules

(3 layers in actual)

Restructuring on Design level

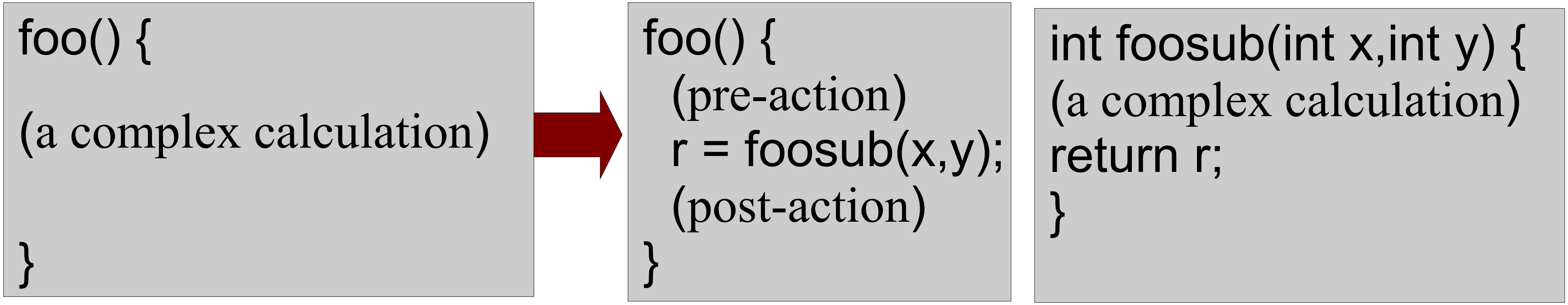
Besides to build system/w acc. from the scratch, we try to restructure current systems using component technologies.

Restructuring on code level is called *refactoring*, widely applied in many development processes.

Note: it only changes structure, never change its function/malfunction

Ex: extract method

Aim : specify calculation clearly / improve possibility for enhancement



```
foo() {

(a complex calculation)


}
```

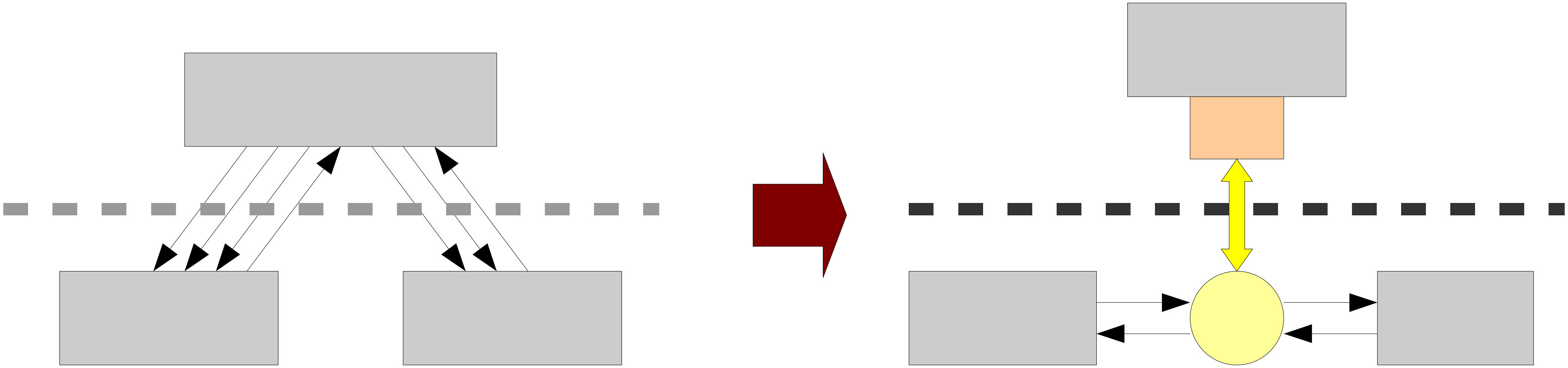
```
foo() {
  (pre-action)
  r = foosub(x,y);
  (post-action)
}
```

```
int foosub(int x,int y) {
(a complex calculation)
return r;
}
```

Restructuring on Design level

We need to reconstruct info. sys. in design level in order to provide accountability because

- legacy systems might not be properly layered
- legacy systems might not have clear interfaces



Style / amount / frequency of communication might be clues to decide layers / interfaces.

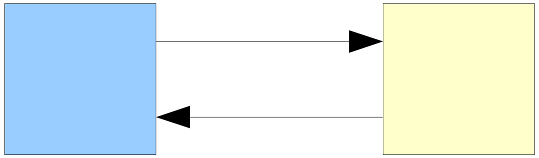
Communication Category

Communication styles are categorized as follows:

(a) One-to-one, synchronous :

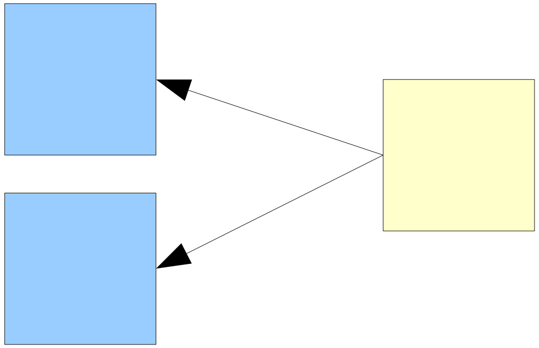
Request/response pair

DB query/resultset pair



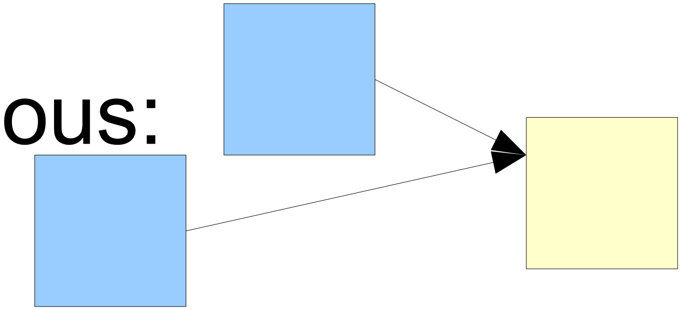
(b) One-to-many,synchronous:

shared data (blackboard) access



(c) One-to-many, asynchronous:

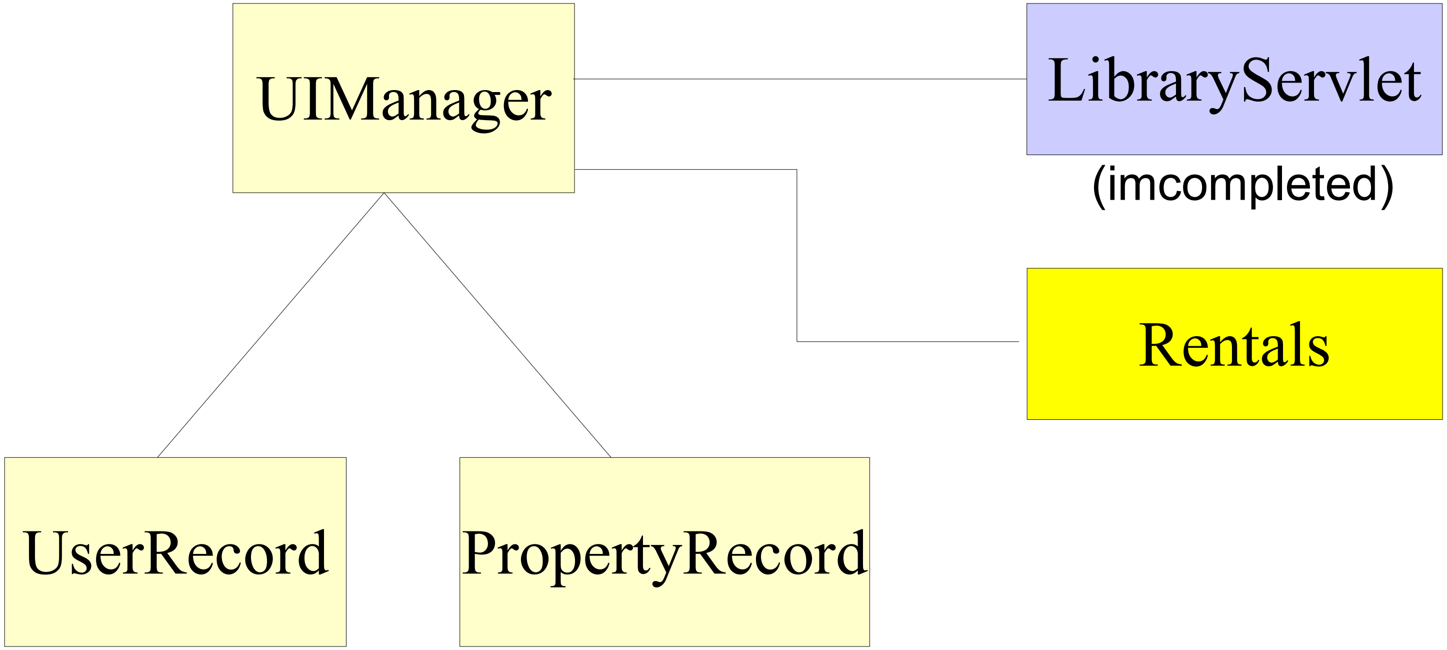
logging



A Case Study

Small library systems in our laboratory

Before : stand-alone, fixed GUI, integrated DB
After: accessible through WEB, distributed DB (final goal)



(imcompleted)

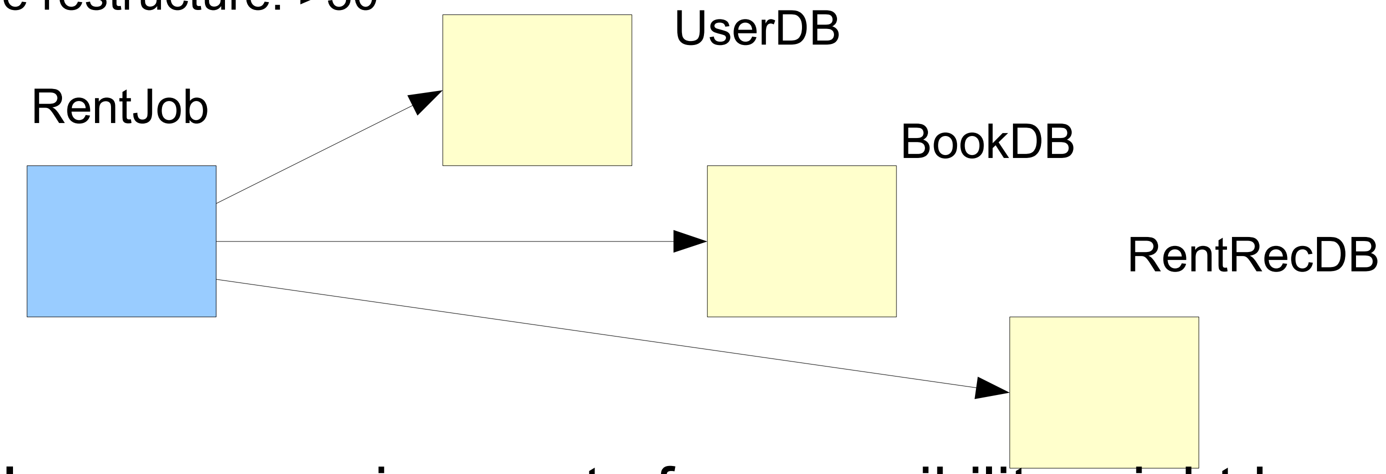
request/response with HTTP

Abstraction of transaction

Communications in Example

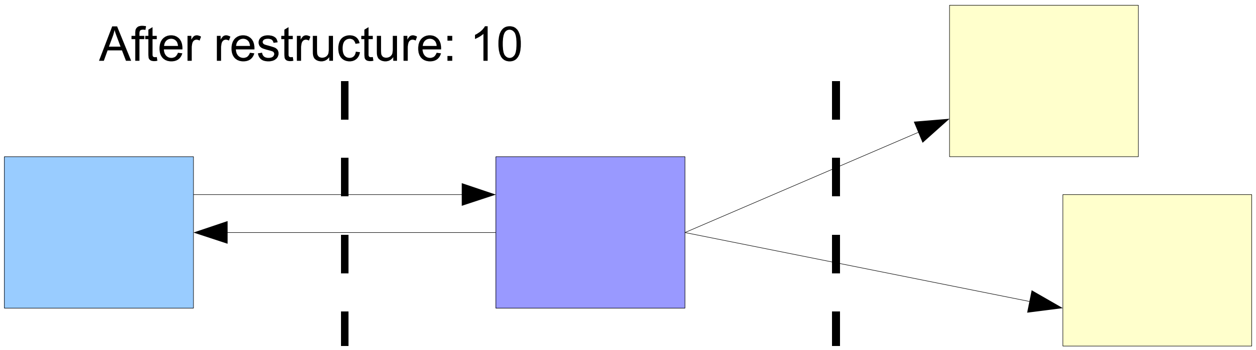
Number of one-to-many sync. comm. is large.

Before restructure: >30



Improper assignment of responsibility might be a cause of increase of comm. So we restructure them as follows:

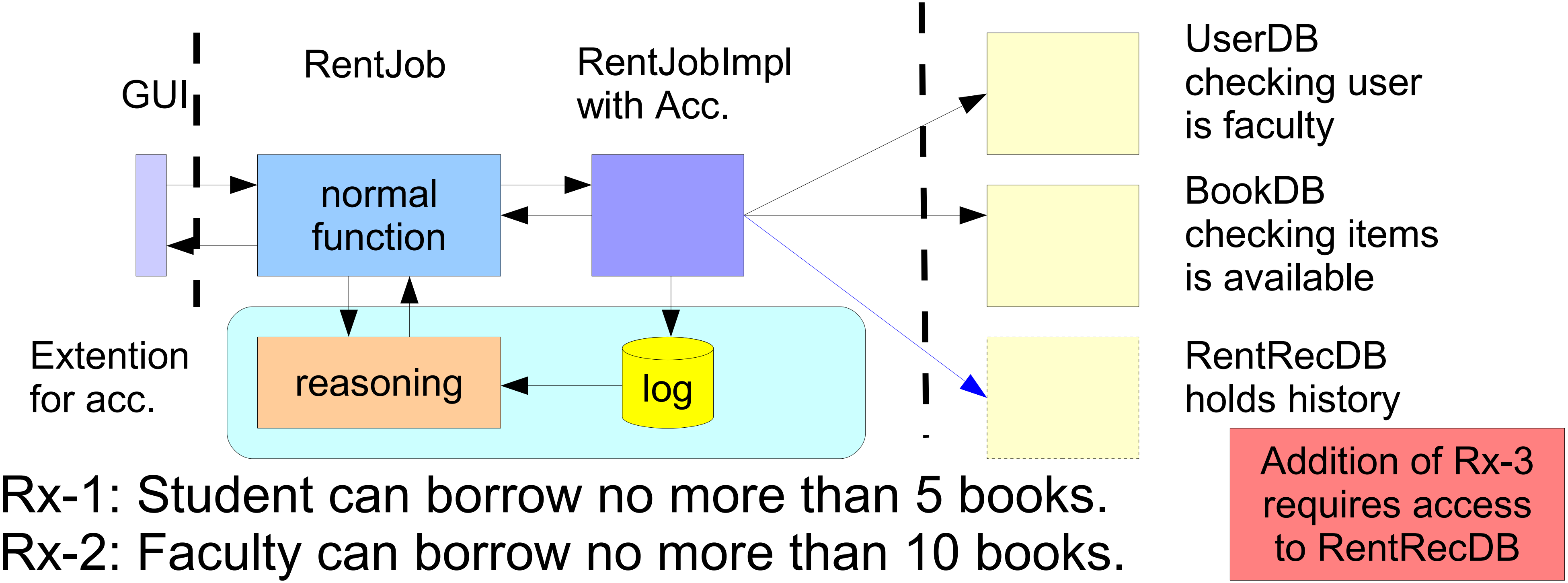
After restructure: 10



Some request are not necessary to access lower layer, but can make responses in middle.

Mechanisms for Accountability

- Reasoning might be introduced in middle layer.
- Implemented by replacing some components with those have accountability-related features.



Rx-1: Student can borrow no more than 5 books.
Rx-2: Faculty can borrow no more than 10 books.
Rx-3: Person who already borrow some books cannot exceed the limit incl. # of books he/she has not yet returned.

Current Status/Summary

We are engaged to establish a development process for info. sys. with accountability using component technologies.

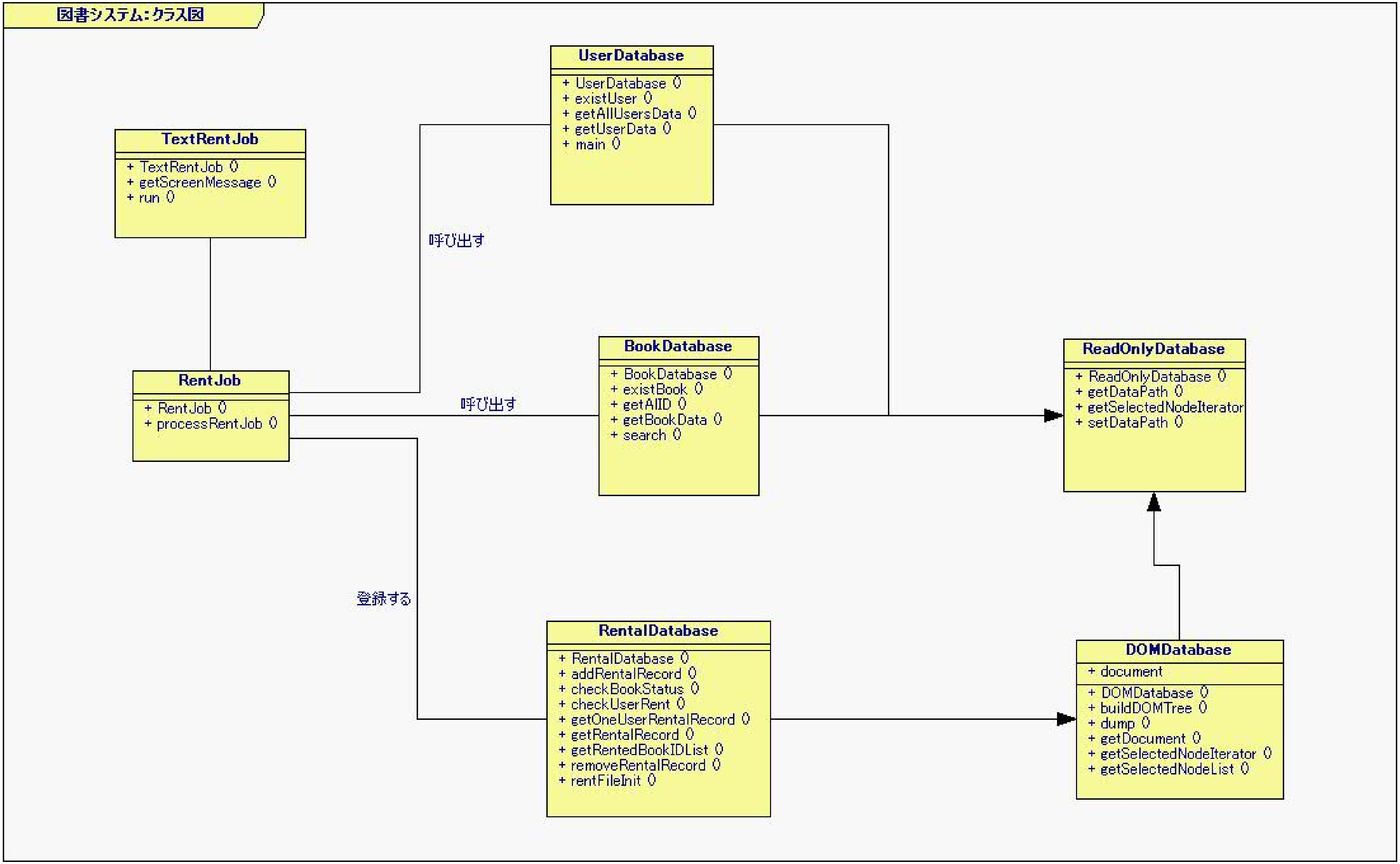
Top-down approach :
extract classes from expression of laws(rules) and use-cases, realize them with components

Prototype of a mapping from query to rule is built and evaluation is in progress.

Bottom-up approach :
extract interfaces from style/amount of interaction, restructure systems into layers, build with comp.

Rules for extracting interfaces are defined and polished through some small systems (incl. mini-library.)

Appendix : Class Diagram for mini library system


図書システム:クラス図
UserDatabase
+ UserDatabase ()
+ existUser ()
+ getAllUsersData ()
+ getUserData ()
+ main ()
TextRentJob
+ TextRentJob ()
+ getScreenMessage ()
+ run ()
呼び出す
RentJob
+ RentJob ()
+ processRentJob ()
呼び出す
BookDatabase
+ BookDatabase ()
+ existBook ()
+ getAllID ()
+ getBookData ()
+ search ()
ReadOnlyDatabase
+ ReadOnlyDatabase ()
+ getDataPath ()
+ getSelectedNodeIterator
+ setDataPath ()
登録する
RentalDatabase
+ RentalDatabase ()
+ addRentalRecord ()
+ checkBookStatus ()
+ checkUserRent ()
+ getOneUserRentalRecord ()
+ getRentalRecord ()
+ getRentedBookIDList ()
+ removeRentalRecord ()
+ rentFileInit ()
DOMDatabase
+ document
+ DOMDatabase ()
+ buildDOMTree ()
+ dump ()
+ getDocument ()
+ getSelectedNodeIterator ()
+ getSelectedNodeList ()