Data access control and measure in the development of web-based health insurance systems

Zhen Jiang
Computer Science Department
Information Assurance Center
West Chester University
E-mail: zjiang@wcupa.edu
Outline

- Introduction
- Problem
- Our approaches
- Conclusion
Introduction

- Policies
- Why?
- Unified Modeling Language (UML) design
- Data attribute
- Data access
  - All the access of a certain data
  - All the access by using member functions of an object (effect area)
- Control of data access
  - Information leakage
  - Fully support of access requirement
  - Redundant access path
Problem

- Control on member functions and relationships between classes
- Control on basic relationships (generalization, aggregation, and association) and their usage in our insurance systems
- Control in design phrase
Our approaches

- Design framework of web-based health insurance systems
  - Quickly catch the complex relations in real world and map to UML diagram
  - Simple system, easy to analyze
- Measure the (possible) accesses of a certain class
  - To check if the requirement of all those accesses can be met (completeness check)
  - To check if the access is feasible (the permission is assigned)
  - To avoid adding redundant access path
  - To provide information for future access conflict analysis
Our approaches

- Design of our health insurance systems
Our approaches

- Generalization, association, and aggregation
  - Association is basic relation.
  - Class A is a super class of class B if and only if any object of class B can also play the role as an object of class A.
  - Class A, as the whole class, has a whole-part relationship with class B if and only if any object of class B belonging to an object of class A has a member function involved in the action of a member function of the later one as a part of that.
  - Association class of two classes A and B will be consider a part of relationships between classes A and B.
  - All-to-all aggregation is used for multilayer implementation.
Our approaches

- Discovery:
  - It could be much easier for us to draw a UML class diagram if we start from the analysis of relationships of components.
  - The more natural the relationships between UML classes, the easier we read and understand the UML design, the easier and faster the development and maintenance of such a system.
Our approaches

- Data access by member functions
  - Data access defined in the same class
  - Data access from other class
- Rules for assignment of call permission of member functions propagating along all kinds of relations.
  - Rule 1: For any two classes A and B, if A is subclass of B, \( P(A, B) = \text{true} \).
  - Rule 2: For any two classes A and B, if A is association related to B, \( P(A, B) = \text{true} \).
  - Rule 3: For any two classes A and B, if A is whole class of B, \( P(A, B) = \text{true} \).
  - Rule 4: For any association class A of two association related classes B and C, \( P(A, B) = P(A,C) = P(B,A) = P(C,A) = \text{true} \).
  - Rule 5: In a multi-layer system, if class A is in the upper layer and class B is in the lower layer, \( P(A,B) = \text{true} \).
Our approaches

- Permission assignment collected in set of access permission (SOAP)
  - Initially, SOAP(C) = {C} for each class C.
  - Based on rules 1-5, find Re(C) = {X | P(C, X) = true}.
  - Repeat SOAP(C) = SOAP(C) ∪ {Y | Y ∈ SOAP(X) ∧ X ∈ Re(C)} in each round until there is no change of any SOAP.
### Our approaches

<table>
<thead>
<tr>
<th>classes</th>
<th>Re</th>
<th>SOAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initially</td>
<td>Roud 1</td>
</tr>
<tr>
<td>_txtBox(Tx)</td>
<td>W, H, P, I, M, D, Ti</td>
<td>Tx</td>
</tr>
<tr>
<td>Single (S)</td>
<td>Tx, H, P, I, M, D, Ti</td>
<td>S</td>
</tr>
<tr>
<td>Table (Ta)</td>
<td>Tx, H, P, I, M, D, Ti</td>
<td>Ta</td>
</tr>
<tr>
<td>Hospital (H)</td>
<td>P, T</td>
<td>H</td>
</tr>
<tr>
<td>Patient (P)</td>
<td>H, T</td>
<td>P</td>
</tr>
<tr>
<td>Ins_Company (I)</td>
<td>M, Ti, T</td>
<td>I</td>
</tr>
<tr>
<td>Member (M)</td>
<td>P, I, D, Ti, T</td>
<td>M</td>
</tr>
<tr>
<td>Dependent (D)</td>
<td>P, M, T</td>
<td>D</td>
</tr>
<tr>
<td>Time (T)</td>
<td>I, M, T</td>
<td>Ti</td>
</tr>
<tr>
<td>Transaction (T)</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>
Our approaches

- For each class C in our system, we provide a new measure $U(C) = \{ X | C \in SOPA(X) \}$ to see all the places in which the member function(s) of class C could be used (directly or indirectly).
Our approaches

<table>
<thead>
<tr>
<th>Classes</th>
<th>SOAP</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebItem (W)</td>
<td>W, H, P, I, M, D, Ti, T</td>
<td>W, Tx, B, Pi, S, Ta</td>
</tr>
<tr>
<td>_txtBox (Tx)</td>
<td>Tx, W, H, P, I, M, D, Ti, T</td>
<td>Tx, S, T</td>
</tr>
<tr>
<td>Button (B)</td>
<td>B, W, H, P, I, M, D, Ti, T</td>
<td>B</td>
</tr>
<tr>
<td>Picture (Pi)</td>
<td>Pi, W, H, P, I, M, D, Ti, T</td>
<td>Pi</td>
</tr>
<tr>
<td>Single (S)</td>
<td>S, Tx, H, P, I, M, D, Ti, W, T</td>
<td>S</td>
</tr>
<tr>
<td>Table (Ta)</td>
<td>Ta, Tx, H, P, I, M, D, Ti, W, T</td>
<td>Ta</td>
</tr>
<tr>
<td>Hospital (H)</td>
<td>H, P, T</td>
<td>W, Tx, B, Pi, S, Ta, H, P, I, M, D, Ti</td>
</tr>
<tr>
<td>Patient (P)</td>
<td>P, H, T</td>
<td>W, Tx, B, Pi, S, Ta, H, P, I, M, D, Ti</td>
</tr>
<tr>
<td>Ins_Company(I)</td>
<td>I, M, Ti, T, P, D, H</td>
<td>W, Tx, B, Pi, S, Ta, I, M, D, Ti</td>
</tr>
<tr>
<td>Member (M)</td>
<td>M, P, I, D, Ti, T, H</td>
<td>W, Tx, B, Pi, S, Ta, I, M, D, Ti</td>
</tr>
<tr>
<td>Dependent (D)</td>
<td>D, P, M, T, H, I, Ti</td>
<td>W, Tx, B, Pi, S, Ta, I, M, D, Ti</td>
</tr>
<tr>
<td>Time (Ti)</td>
<td>Ti, I, M, T, P, D, H</td>
<td>W, Tx, B, Pi, S, Ta, I, M, D, Ti</td>
</tr>
<tr>
<td>Transaction (T)</td>
<td>T</td>
<td>W, Tx, B, Pi, S, Ta, H, P, I, M, D, Ti, T</td>
</tr>
</tbody>
</table>
Our approaches

- No redundant access path
- All access will be ensured in design phrase
- Completeness check process to build assurance of data access of class C.
  - For any existing class \( X \in U(C) \), check if class \( X \) needs to call any member function \( f \) of \( C \) although the permission has been assigned.
  - When such \( X \) is not found in SOAP(C), only check if \( C \)'s member functions support the requirement of \( X \). If not, redesign the member functions of \( C \).
  - When such \( X \) is also found in SOAP(C), both \( X \) and \( C \) needs check at the same time and may need redesign if there is conflict or inconsistency among their member functions.
Conclusion

- Control and measure in design phrase
- A control solution from the side of computer science
- Impact analysis