Abstracting Business Level UML Diagrams from Web Applications

P. Tramontana

U. De Carlini, A.R. Fasolino

Dipartimento di Informatica e Sistemistica

University of Naples Federico II, Italy

G.A. Di Lucca

RCOST – Research Centre on Software Technology

University of Sannio, Benevento, Italy

Web Applications (WA): problems and open issues

- World Wide Web is evolving: every year new technologies and architectures are realized and WA potentiality and performance rapidly grow.

- Under the pressure of a short time-to-market and an extremely high competition, WA are realized with a limited quality:

- WA developed without a disciplined process
- Poor, inadequate, incomplete documentation
- Disordered architecture

Web Applications (WA): problems and open issues

⇒ Maintenance, Reengineering and Migration are very critical tasks for WA

⇒ Object Oriented Conceptual Models are a fundamental starting point for these tasks

Object Oriented Conceptual Models

⇒In this work, methodologies will be presented to abstract Object Oriented Conceptual Models of a Web Application.

✓ Business Level UML Class Diagrams
✓ UML Use case diagrams
✓ UML Sequence diagrams

Business Level UML Class Diagrams

- Business level UML class diagrams will describe the relevant conceptual components (*business objects*) from the domain of the problem addressed by the WA and their mutual relationships

Recovering a Business Level Class Diagram

We propose a Reverse Engineering process in three main steps:

1) Identification of candidate classes and their attributes;
2) Association of methods to candidate classes;
3) Identification of relationships between classes.

1) Identification of candidate classes

- Searching for groups of logically related data making up the state of objects.

⇒Looking into the source code for language mechanisms that allow grouping of related data implementing a relevant concept, either from the domain of the application or from domain of the solution

Identification of relevant groups of data

- Groups of data in input/output operations (HTML forms, HTML tables);
- Groups of data in read/write operations (Arrays, Collections);
- Groups of data in database operations (Recordsets)
- Groups of data passed throw distinct pages (Querystrings);
- Instance of Classes

Synonyms and Homonyms Analysis

- *Synonyms* are identifiers with different names but the same meaning
- *Homonyms are* identifiers with the same name but different meanings

⇒*Synonym* identifiers must be assigned with the same unique name.

⇒*Homonym* identifiers must be associated with distinct names.

During *synonyms & homonyms* analysis, a meaningful name is assigned to each data item

An automatic procedure to propose a set of candidate classes

- Data groups are oredered with two heuristic rules:

1) the more the references of a same data group in the code, the greater the likelihood that it represents a meaningful concept;
2) data groups with a small size may represent more simple and atomic concepts than larger groups, and larger groups may represent more complex concepts made up of joined smaller groups.

Example

	Group name	Composition	#
➔	G1	a, b, c	4
➔	G2	d, e, f	3
➔	G3	a, b, c, d, e, f, g	2
➔	G4	a, b, c, m, n	2
➔	G5	a, b, q, r	2
➔	G6	d, f, s, t, u	1

	CAND
C1	a, b, c, g, m, n
C2	d, e, f, g
C3	a, b, q, r
C4	d, f, s, t, u

Procedure *Produce_Candidate_Objects (in:* GList; out: CAND);

```
BEGIN
OrdList = SORT (Glist, Descending on $N_{ref}(g)$AND Ascending on
   Card($g$) );
CAND = ∅;
WHILE (OrdList ≠ ∅ OR ($\cup_i$ a ∈ $C_i$ ≡ $\cup_i$ a ∈ $g_i$ ))DO
   $h$=TOP(OrdList);
   IF (∃ a∈$h$: a∉$C$ ∀ $C$ ∈CAND) THEN
        IF (!∃ $C$∈CAND: $C$ ⊆ $h$ THEN
                INSERT(CAND, $h$)
   ELSE
                ∀ $C_i$ ∈ CAND: $C_i$ ⊆ $h$ DO
                        BEGIN
                        $k$ = $h$ - $\cup_i$ $C_i$;
                        ADD($C$, $k$);
                        END
        END IF
   END IF
   REMOVE (OrdList, $g$);
END WHILE
END
```

2) Associating methods to candidate classes

- Possible functional units to be considered should include web pages, functions, script blocks, depending on the requested degree of granularity.

⇒Our approach proposes to consider physical Web pages (e.g., Server pages, Client pages) as potential methods to be associated with the candidate objects of the WA, rather than inner page components

Coupling between pages and objects

- Measures of coupling between pages and objects are computed based on the accesses of pages to the candidate object. A page accesses a candidate object when it includes instructions that *define* or *use* the value of some object attribute

$\Rightarrow Acc(m, c) = \alpha_{def} * N_{def,m,c} + \alpha_{use} * N_{use,m,c}$

where:

– $\alpha_{def}, \alpha_{use}$: weights associated with define,use access;

- $N_{def,m,c}$: number of definition of candidate c in page m;

- $N_{use,m,c}$: number of uses of candidate c in page m.

Associating pages to candidate classes

⇒ If a page accesses exclusively one object, it will be assigned as a method of that object.

⇒ If a page m accesses more objects, it will be assigned to the candidate class c so that

$$Acc(m , c) = MAX_{c_j \in CAND} [Acc(m , c_j)]$$

The accesses of the page to the other objects can be considered as messages exchanged from the object the page has been assigned with, to the other objects

⇒ Pages that do not make access to any objects will be considered as coordinating modules controlling the executions of other methods

3) Identification of relationships between classes

- Common attributes
  - If two or more classes C_i have a common attribute:
    - ⇒ the attribute is assigned to one of these classes (C_j) and deleted from other classes C_k;
    - ⇒ A relationship will be defined between the class C_j and each class C_k

⇒These relationships will be depicted as UML *association* relationships

3) Identification of relationships between classes

- Pages accessing more than one class
  - ⇒ the accessing page will be assigned with the class it is most highly coupled with.
  - ⇒ A relationship will be defined between the class the page is assigned to, and each remaining class the page accesses.

⇒Also this kind of relationships will be depicted as UML *association* relationships

Recovering Use Case Diagrams

- ⇒ An automatic clustering approach is proposed, in order to group Web pages of a WA into meaningful (highly cohesive) and independent (loosely coupled) clusters
- ⇒ Groups of validated clusters can be associated with potential use cases of the WA, and a use case model can be reconstructed for describing the external behavior offered by the application to the end users.

Example

- ⇒Relationships among use cases may be deduced analyzing the links between corresponding clusters:
  - ⇒If a cluster A is linked to just another cluster B, an <<include>> relationship from use case associated with A to the one associates with B may be proposed;
  - ⇒If a cluster A is linked to more other clusters B_i, a <<extend>> relationship among use case associated with A and use cases associated with B_i may be proposed;

Recovering Sequence Diagrams

- For each use case:
  - we consider the pages belonging to the associated cluster;
  - for each page in the cluster:
    - an object of the class which the page was assigned to as a method is drawn;
    - identify the class which the page was assigned to as a method: for each identified class put an object in the diagram;
    - if a page accesses other objects of classes B_i, an interaction between A and B_i is drawn;

Recovering Sequence Diagrams

- a meaningful name is assigned to each interaction (corresponding to the name of the invoked method), with the list of parameters exchanged between the objects (deduced by analysing the data flow between the Web application interconnected items);
- if an object was assigned to a page including input or output forms, an interaction between the object and an actor is drawn;
- if there is a relation between pages assigned to the same class, it will be considered as a call between methods of the same Class, and therefore a 'Message to self' is drawn on the object life-line.

A case study

- Human Net: a WA designed to support the activities of undergraduated courses offered by a Computer Science Department
- Developed with ASP, Javascript, HTML languages
- 75 server pages, 23 client pages, 1 utility module (7648 LOCs)

Identification of groups of data

- After static analysis of the WA:
  - 128 references to data groups (485 data items)
- After Synonyms and Homonyms Analysis:
  - 43 different data groups (26 different data items)

Data Groups

An excerpt of the 43 different Data Groups obtained after Synonyms and Homonyms Analysis

	Data groups	#
C1	Student name, Student surname, Student code, Student email, Student phone number, Student password	14
C2	Teacher name, Teacher surname, Teacher email, Teacher phone number, Teacher password, Teacher code	11
C3	Exam date, Exam time, Exam classroom	10
C4	Student name, Student surname, Student code, Student email, Student phone number	8
C5	Tutoring date, Tutoring start time, Tutoring end time, Course code, Course name	7
C6	Student code	6
C7	Course code, Course name, Course academic year	5
C8	Course code	4
C9	Course code, Course name	4
C10	Teacher name, Teacher surname, Teacher email, Teacher phone number, Teacher password	4
C11	Course code, Course academic year	3

Candidate classes

8 Candidate Classes obtained after the execution of the algorithm *`Produce Candidate Objects.`*

Candidate classes and corresponding attributes	
Student	(Student name, Student surname, Student code, Student email, Student phone number, Student password)
Teacher	(Teacher name, Teacher surname, Teacher email, Teacher phone number, Teacher password, Teacher code)
Exam Session	(Exam date, Exam time, Exam classroom)
Tutoring	(Tutoring date, Tutoring start time, Tutoring end time, Course code, Course name)
Course	(Course code, Course name, Course academic year)
Tutoring Request	(Student name, Student surname, Student code, Tutoring request date)
News	(Course code, News text, News number, News date, Teacher code)
Exam Reservation	(Student code, Student name, Student surname, Course code, Exam date, Exam reservation date)

Relationships between classes

- 7 Common attributes (*Student name, Student surname, Student code, Teacher code, Exam date, Course code, Course name)*

⇒ 7 relationships were defined

- nn pages access more than one object

⇒ 11 relationships were defined

Associating methods to candidate classes

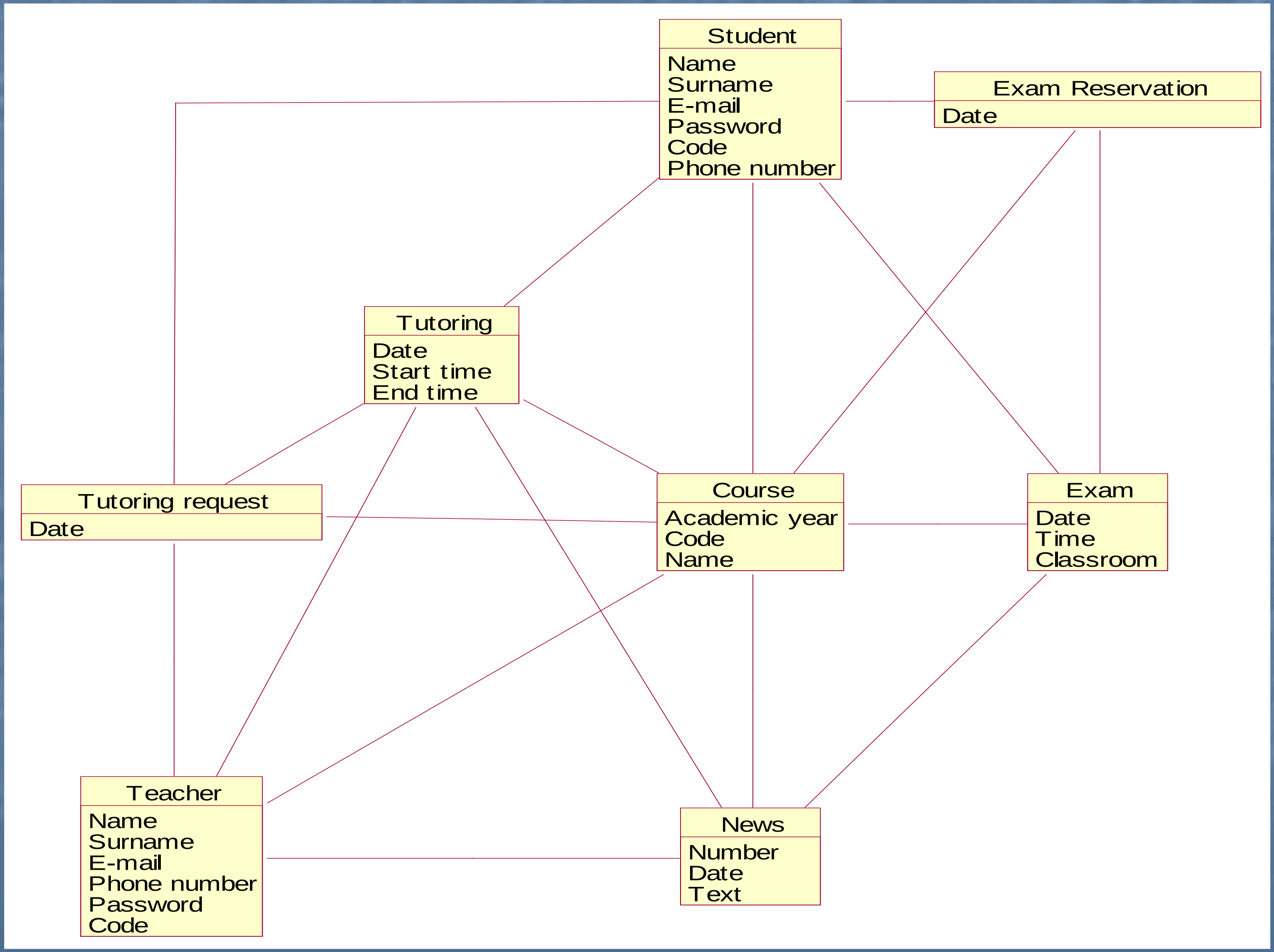
- 20 pages did not reference any data group
- The remaining 78 pages were assigned to the candidate class that maximized the value of
  - Acc (m, c)= $\alpha_{def} * N_{def,m,c} + \alpha_{def} * N_{use,m,c}$

⇒ Several values for α_{def} and α_{use} were tried.

⇒ The best results were achieved for

$\alpha_{def} = 1$, $\alpha_{use} = 0{,}6$

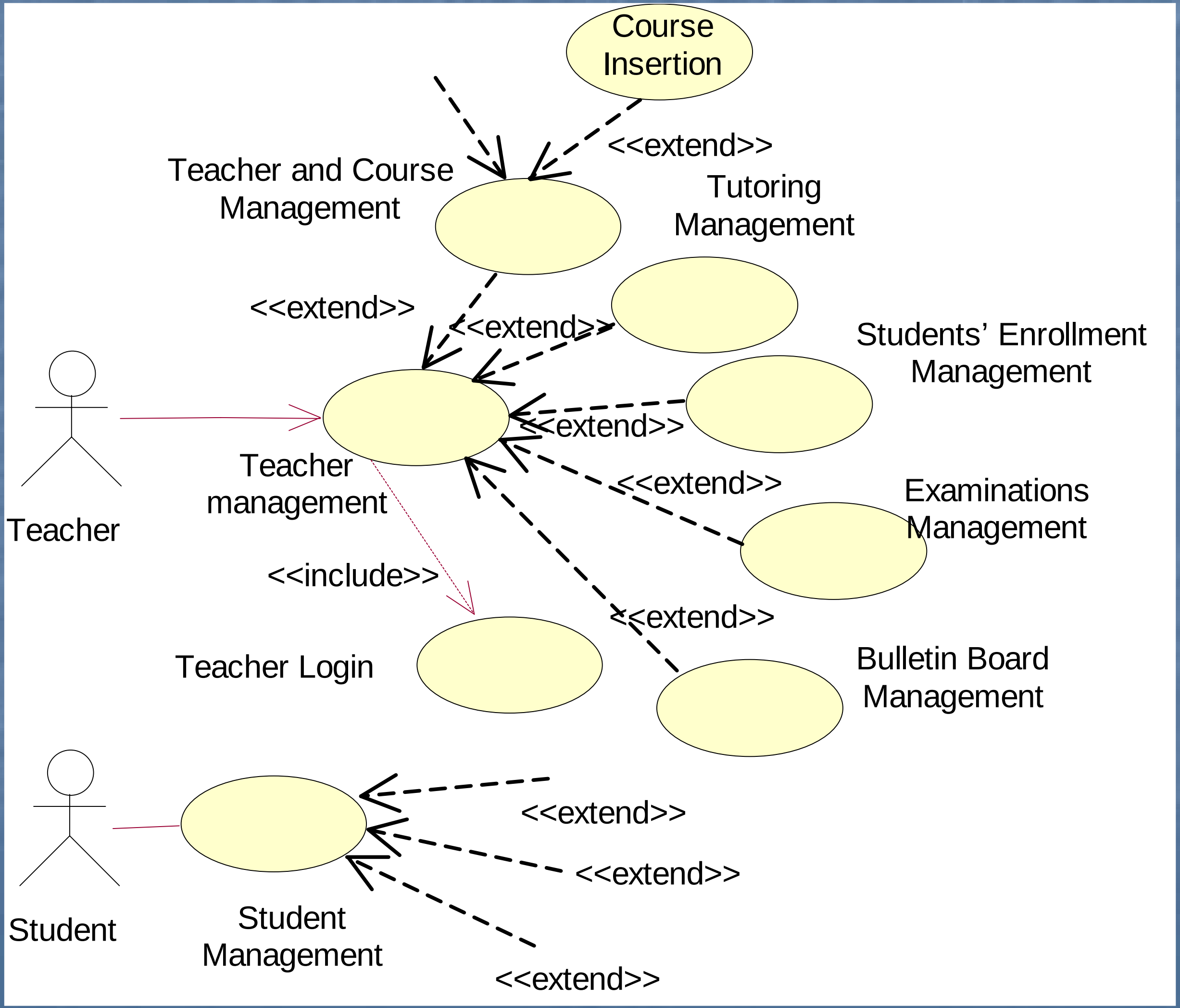
Business Level UML Class Diagram



Abstracting UML Use Case Diagram

- Automatic clustering approach was applied
  - 44 valid clusters were recovered
  - Clusters were associated to use cases
  - <<extend>> and <<include>> relationships were deduced

An excerpt of the UML Use Case Diagram

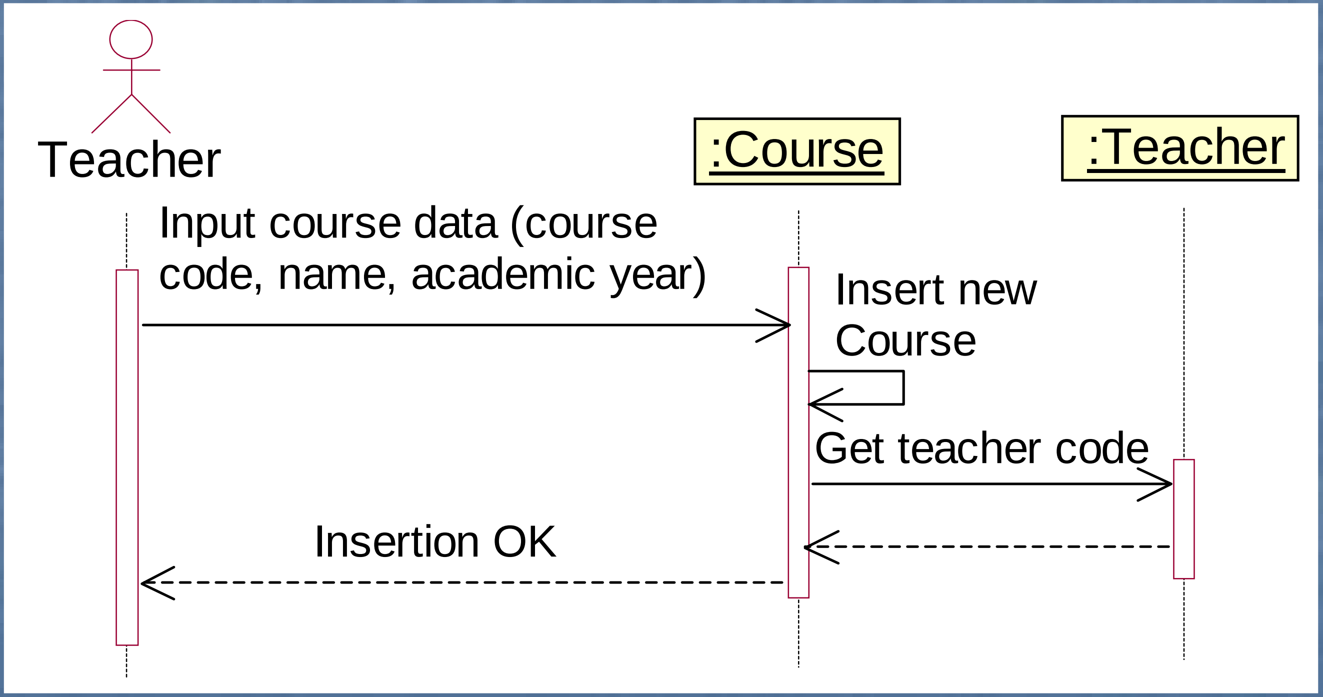


Abstracting UML Sequence Diagram

- For each use case, an UML Sequence Diagram were traced, applying the heuristic rules presented

UML Sequence Diagram

Sequence diagram for use case *Course Insertion* (realized by a cluster of 3 pages)


Teacher
:Course
:Teacher
Input course data (course code, name, academic year)
Insert new Course
Get teacher code
Insertion OK