UCVSC: A Formal Approach to UML Class Diagram Online Verification Based on Situation Calculus

Li Tan*, Zongyuan Yang, Jinkui Xie

* = presenter

ICCIT 2009, Seoul, Korea, Nov. 24-26, 2009

Department of Computer Science and Technology,
East China Normal University,
Shanghai, 200241, China

http://www.cs.ecnu.edu.cn/

Supported by National Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20060269002, Natural Science Fund of Shanghai Municipality under Grant No. 09ZR1409500 and Key Project in Basic Research of Science and Technology Commission of Shanghai Municipality under Grant No. 09JC1405000.

Additional Acknowledgements for this presentation

- Other students involved:
  Yefei Zhao, Qiang Liu and Kangle Cui

- Special thanks to:
  Hugo/RT project lead by Alexander Knapp at LMU, Munich, Germany:

  Hugo/RT is a UML model translator for model checking, theorem proving, and code generation: A UML model containing active classes with state machines, collaborations, interactions, and OCL constraints can be translated into the system languages of the real-time model checker UPPAAL, the on-the-fly model checker SPIN, the system language of the theorem prover KIV, and into Java and SystemC code.

  See also: http://www.pst.ifi.lmu.de/projekte/hugo/

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

Overview of the problem of UML informalism

- Due to the informal graphical notation of UML, flaws cannot be found in the design phase but in the execution phase, which may cost a lot. (50% T_{total} by software testing)
  - To provide UML precise formal semantics is in the spotlight
  - Formal methods such as formal specification and formal verification
  - Incompleteness of requirements, incorrectness of representation and inconsistency of different understanding towards system design can be eliminated (“ 3 I ” Principles)
- UML class diagram is indispensable and worth being verified
  - UML class diagram specifies the structure of a system statically
  - The information it contains is vital and indispensable to the whole design process

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

How situation calculus Deals with These Problems

- With respect to all aspects in UML class diagram, the strength of reasoning about actions and describing the state of the world in situation calculus can apply to represent them appropriately.
  - Though statically deployed as a whole, UML class diagram also has some dynamic features locally such as generalizations, dependencies and associations, the three main relationships between classes, which is just situation calculus good at
  - The static ingredient in UML class diagram can be described by situation element in a first-order way easily

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

UML vs. XMI

- From the initiate idea of introducer, XMI (XML Metadata Interchange) is a framework for defining, interchanging, manipulating and integrating XML data and objects.
  - Used for integration of tools, applications, repositories, data warehouses (device independence of UML modelling)
  - Typically used as interchange format for UML models
- Tool supports for transformation from UML to XMI
  - Rational Rose, Poseidon For UML, ArgoUML, Enterprise Architect
- Why choose ArgoUML?
  - Thinness (10.6MB for installed files which exceeds the other three)
  - Relatively smaller size of XMI file generated (easier to parse)

The ABC's of situation calculus

- Introduced by John McCathy in 1963, situation calculus has been widely applied in Artificial Intelligence related research and other fields.
  - A dialect of logic language
  - Mostly used in dynamic domain modeling
- Key concepts in situation calculus
  - Action, Situation and Fluent
    - ◊ An action represents any possible change to the world
    - ◊ A situation represents a possible world history, simply a sequence of actions
    - ◊ A fluent represents a relation or a function whose truth values varies from one situation to the next, called relational fluent or functional fluent respectively
  - Two predefined binary symbols: Function do and Predicate Poss
    - ◊ do: Action $\times$ Situation $\rightarrow$ Situation eg.: do(a, s)
    - ◊ Poss: Action $\times$ Situation eg.: Poss(a, s)

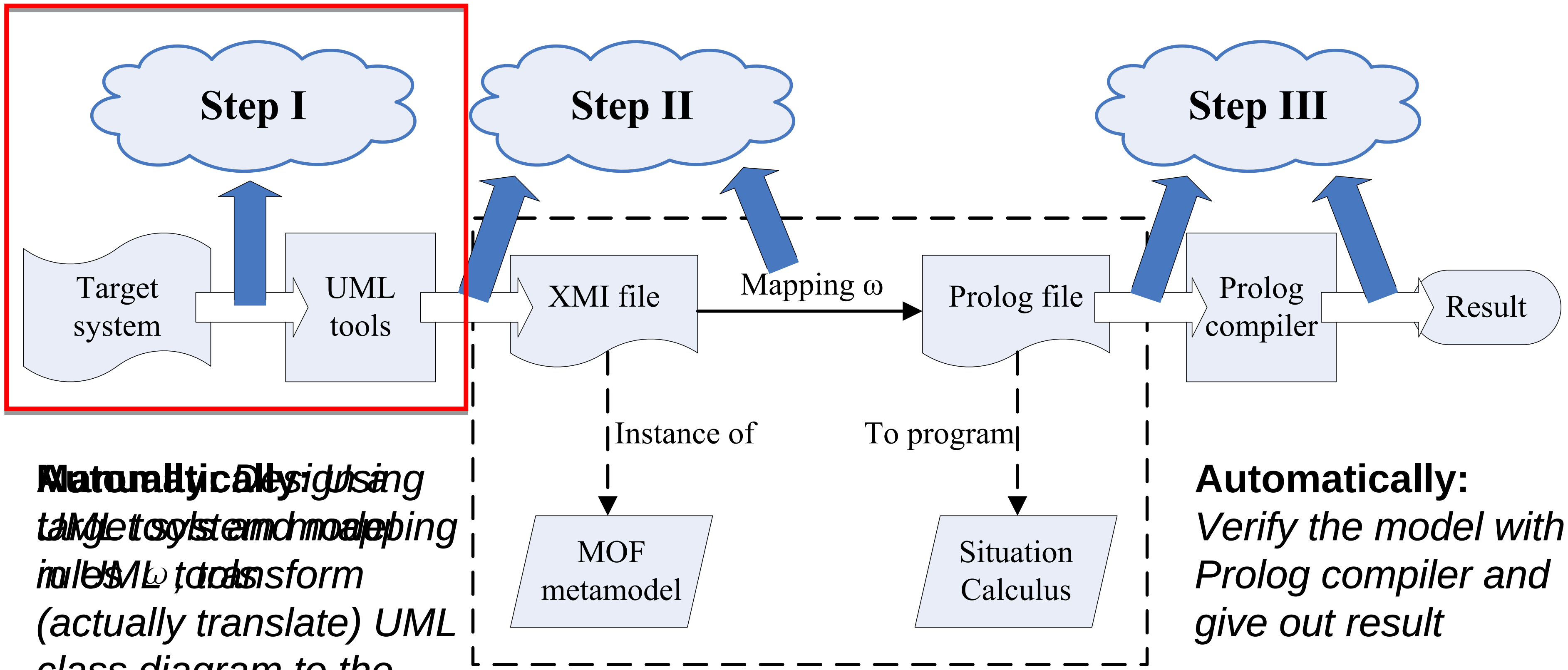
Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

Outline

UML Class diagram online Verification based on Situation Calculus

- To describe it more directly, the overall architecture of our prototype verification system, UCVSC, is shown as follows:



Manually: *Designing target system and mapping rules ω*

Automatically: *Using UML tools and model in UML tools, to transform (actually translate) UML class diagram to the Prolog script in situation calculus.*

Automatically: *Verify the model with Prolog compiler and give out result*

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

An Academic System in UML Class Diagram and Formalization of It

- As a case study, we show an academic system in university in terms of UML class diagram and formalize it by situation calculus as follows:

  Definition 1. A formal structure of a UML class diagram in situation calculus is

  $$\chi_c = \langle Cl, As, G, Ag, Co, D \rangle, \text{ where}$$

  Cl: a finite set of classes, // ‘Cl’ stands for ‘Class’

  As: $C \leftrightarrow C$, a bijection between two classes, // ‘As’: ‘Association’

  G: $C \rightarrow C$, an injection from a child class to its parent class, // ‘G’: ‘Generalization’

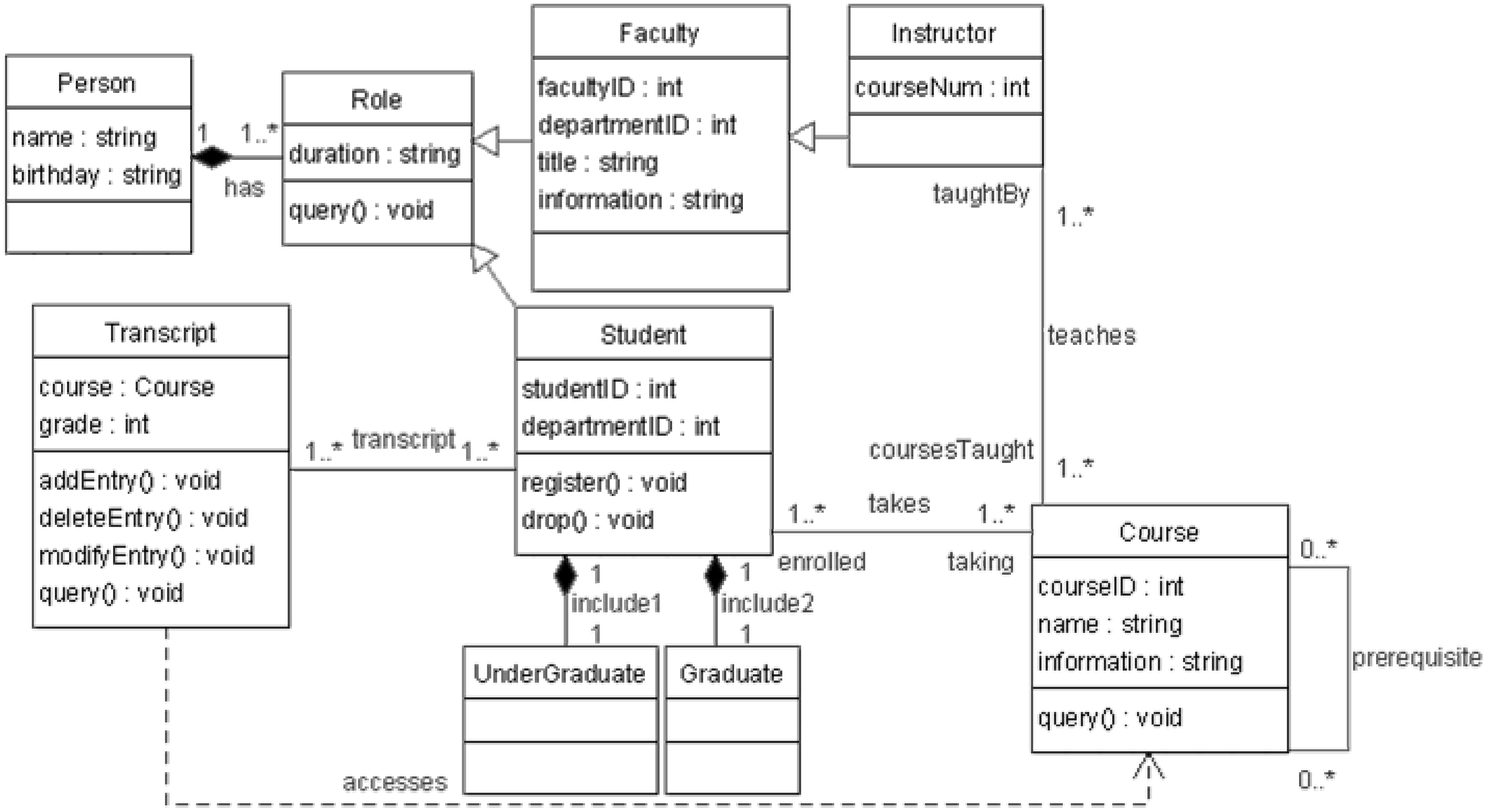
  Ag: $C \rightarrow C$, an injection from a part class to its shared aggregation class, // ‘Ag’: ‘Aggregation’

  Co: $C \rightarrow C$, an injection from a part class to its composite aggregation class, // ‘Co’: ‘Composite’

  D: $C \rightarrow C$, an injection from a friend class to its independent class. // ‘D’: ‘Dependency’

An Academic System in UML Class Diagram and Formalization of It (Cont.)

- The academic system model in UML class diagram is below:



An Academic System in UML Class Diagram and Formalization of It (Cont.)

- Mapping mechanism definition between UML class diagram and situation calculus are as follows:

Definition 2. Mapping from UML class diagram to situation calculus

UML class diagram	situation calculus
Class	Functional Fluent
Association	Relational Fluent/Action
Generalization	Action
Aggregation	Action
Composition	Action
Dependency	Action

Definition 3. A formal model transformation rules from basic elements of UML class diagram to Prolog script which implements situation calculus can be defined in the follow table:

An Academic System in UML Class Diagram and Formalization of It (Cont.)

- Transformation rules definition among UML class diagram, XMI and Prolog script (See examples as follows):
  - Class ‘Faculty’ (Elements in UML class diagram in plain text)

    equals UML:Class name = ‘Faculty’ (XMI script)

    equals Class(Faculty, s) (Prolog script)
  - Association ‘takes’ between Class Student and Course (UML class D)

    equals UML:Association name = ‘takes’
    UML:AssociationEnd.participant (XMI script)

    equals takes(Student, Course, s) (Prolog script)
  - Generalization from ‘Instructor’ to ‘Faculty’ (UML class D)

    equals UML:Generalization name = ‘FtoP’
    UML:Generalization.child (XMI script)

    equals Inherit(Instructor, Faculty, s) (Prolog script)

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

Algorithm for Generalization: transform XMI script to Prolog script

- When parsing the XMI file transformed from UML class diagram of the target system, we propose the following algorithm to formally transform XMI script to situation calculus in Prolog syntax, a compacter format than XMI:
  - Since the others are similar, only the algorithm for the path type of Generalization in UML class diagram is given out
  - This algorithm is not written strictly in terms of standard format, but in the similar syntax of Perl
  - This algorithm is mainly based on iteration, which is determined by the structure of XMI

Algorithm for Generalization: transform XMI script to Prolog script (Cont.)

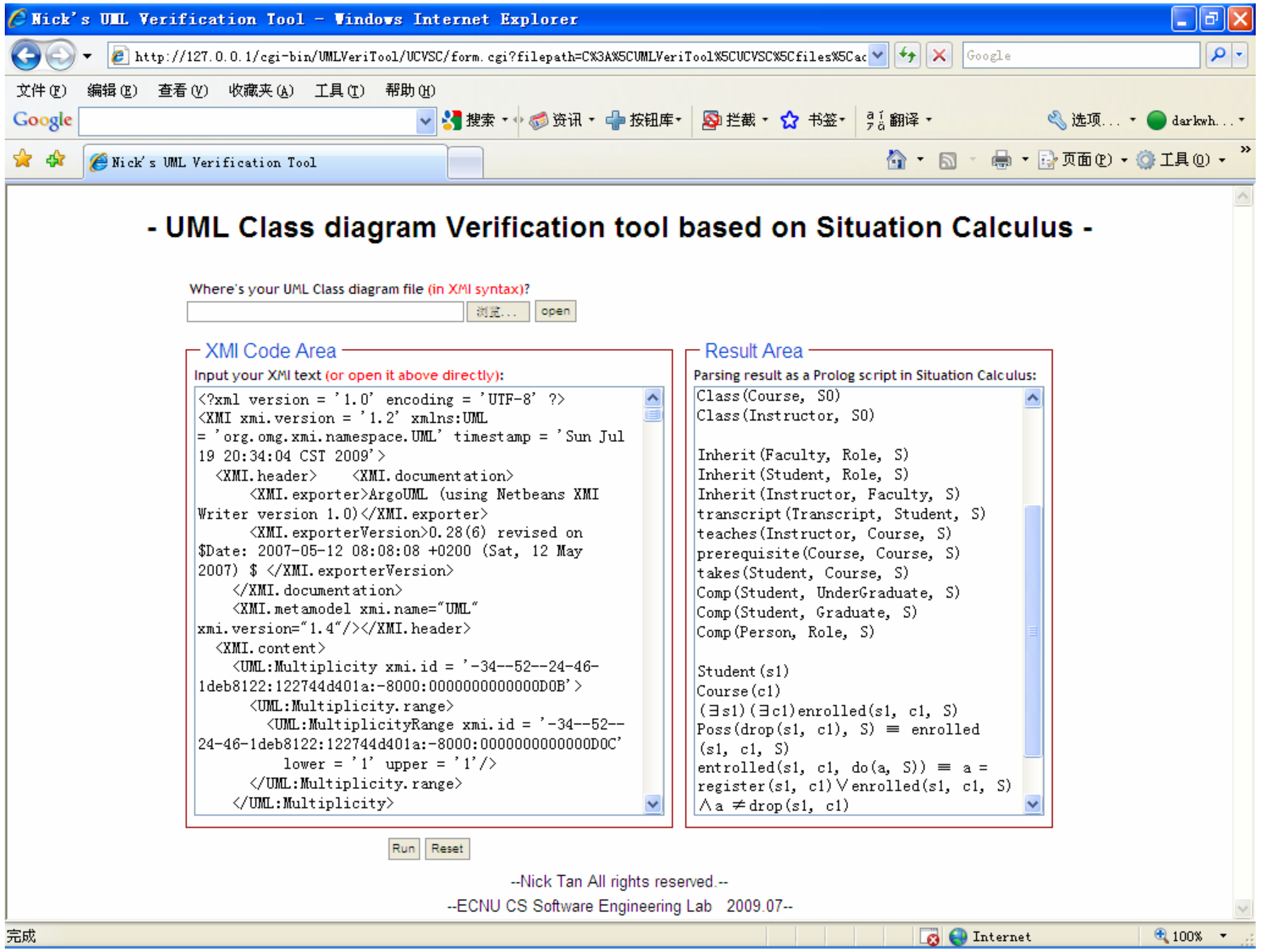
```
Procedure transform_generalization()
1 for each UML.Model in XMI.content in XMI
2    extract attributes.name;
3    gen_exist = false; // global variable
4    for each UML.Class
5        extract attributes.name;
6        print "Class(attributes.name, s)";
7        if(UML.GeneralizableElement.generalization != null)
8            gen_exist = attributes.id;
9            for each UML.Attribute
10               extract attributes.name;
11               for each UML.Operation
12                   extract attributes.name;
13                   if(gen_exist != false)
14                       for each UML.Generization
15                           for each UML.Generization.child
16                               extract child.name; // retrieved by class_id
17                           for each UML.Generization.parent
18                               extract parent.name; // retrieved by class_id
19                           print "Inherit(child.name, parent.name, S)";
end.
```

Implementation of the Prototype System and Verification of the Model

- Based on the structure and function of different parts described above, the prototype system of UML class diagram online verification tool is implemented by Perl
  - Working Principle: After reading the XMI file generated by ArgoUML, this verification tool parses every node, extracts and then translates the information needed to a Prolog script. Finally, our tool will call SWI-Prolog, a Prolog compiler (actually interpreter) to execute this Prolog script and then display the result given out by SWI-Prolog

Implementation of the Prototype System and Verification of the Model (Cont.)

- The following figure shows the user interface of our prototype system:


Nick's UML Verification Tool - Windows Internet Explorer
http://127.0.0.1/cgi-bin/UMLVeriTool/UCVSC/form.cgi?filepath=C%3A%5CUMLVeriTool%5CUCVSC%5Cfiles%5Cac
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
Nick's UML Verification Tool
完成
Internet
100%

- UML Class diagram Verification tool based on Situation Calculus -

Where's your UML Class diagram file (in XMI syntax)?

浏览... open

XMI Code Area

Input your XMI text (or open it above directly):

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML
= 'org.omg.xmi.namespace.UML' timestamp = 'Sun Jul
19 20:34:04 CST 2009'>
  <XMI.header>    <XMI.documentation>
      <XMI.exporter>ArgoUML (using Netbeans XMI
Writer version 1.0)</XMI.exporter>
      <XMI.exporterVersion>0.28(6) revised on
$Date: 2007-05-12 08:08:08 +0200 (Sat, 12 May
2007) $ </XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML"
xmi.version="1.4"/></XMI.header>
  <XMI.content>
    <UML:Multiplicity xmi.id = '-34--52--24-46-
1deb8122:122744d401a:-8000:000000000000D0B'>
      <UML:Multiplicity.range>
        <UML:MultiplicityRange xmi.id = '-34--52--
24-46-1deb8122:122744d401a:-8000:000000000000D0C'
          lower = '1' upper = '1'/>
      </UML:Multiplicity.range>
    </UML:Multiplicity>
```

Result Area

Parsing result as a Prolog script in Situation Calculus:

```
Class(Course, S0)
Class(Instructor, S0)

Inherit(Faculty, Role, S)
Inherit(Student, Role, S)
Inherit(Instructor, Faculty, S)
transcript(Transcript, Student, S)
teaches(Instructor, Course, S)
prerequisite(Course, Course, S)
takes(Student, Course, S)
Comp(Student, UnderGraduate, S)
Comp(Student, Graduate, S)
Comp(Person, Role, S)

Student(s1)
Course(c1)
(∃s1)(∃c1)enrolled(s1, c1, S)
Poss(drop(s1, c1), S) ≡ enrolled
(s1, c1, S)
entrolled(s1, c1, do(a, S)) ≡ a =
register(s1, c1)∨enrolled(s1, c1, S)
∧a ≠drop(s1, c1)
```

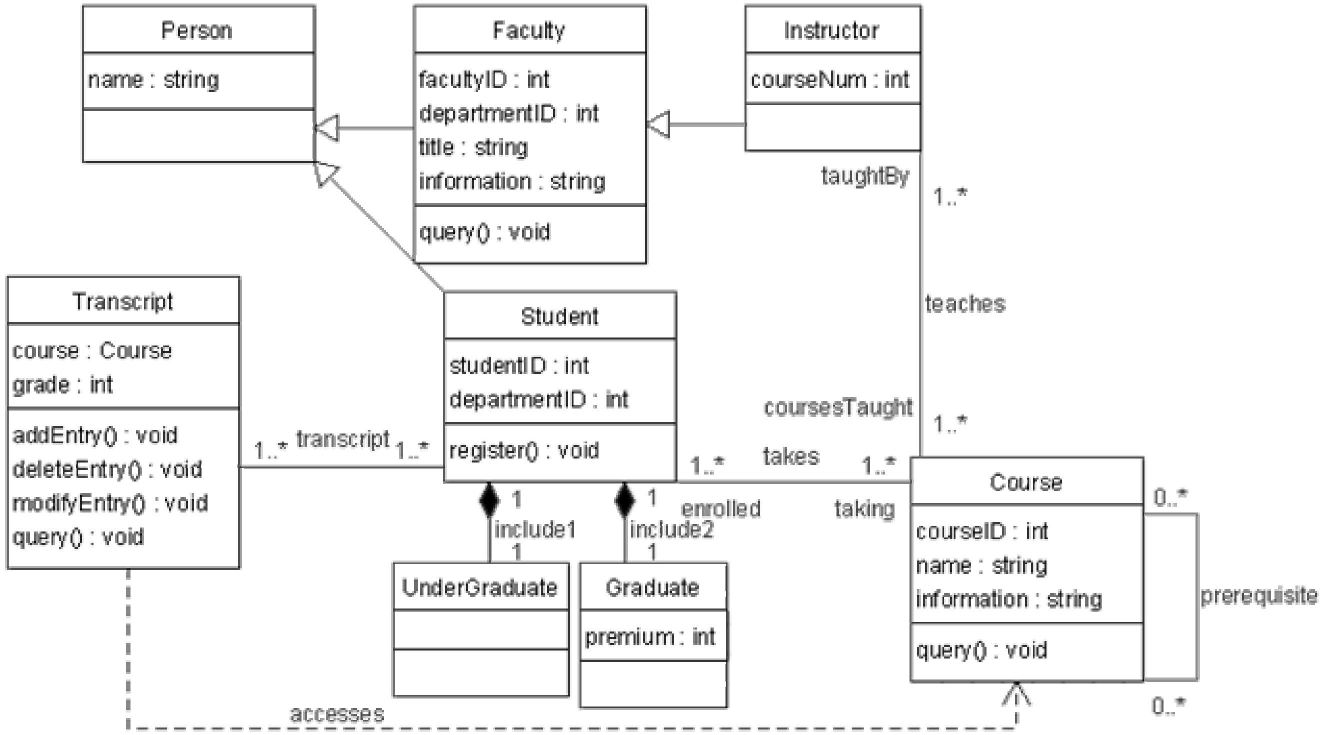
Run Reset

--Nick Tan All rights reserved.--

--ECNU CS Software Engineering Lab 2009.07--

Implementation of the Prototype System and Verification of the Model (Cont.)

- If we use SWI-Prolog to execute the Prolog script generated by our tool, it will pass on the premise that the design of the academic system described as above is correct. What if the design is bad? Now, let’s take a look at another design version of the same academic system in the next figure:



Implementation of the Prototype System and Verification of the Model (Cont.)

- Apparently the difference between 2 designs lies in the deletion of abstract class Role (i.e., class Person becomes the direct parent class of class Faculty and Student).
- Then problem comes. The latter design does not permit a teacher to be a student. Virtually, it’s a commonplace for a teacher to get further study in the same university. Thus, it’s incorrect and unreasonable to delete abstract class Role in the latter figure.
  - Fortunately, with the help of Prolog compiler, the Prolog script as follows cannot pass. // A student and teacher case in Prolog script

```
Course(CS01, S0);
Course(CS04, S0);
…
Student(Nick, S0);
…
take(Nick, CS04, S);
teach(Nick, CS01, S);
```

Agenda

- Overview of the problem of UML informalism
- Why situation calculus?
- Background knowledge (UML vs. XMI, situation calculus)
- Outline of the prototype system, UCVSC
- Formalization of an academic system in UML class diagram
- Implementation and verification
- Prospective routes for improvement

Prospective routes for improvement

- Our contributions and value of work:
  - With the help of UCVSC, the formal online verification tool for UML class diagram we have developed, software architecture engineers who design a software system can find the incorrectness in static structure of this system in advance, rather than to redesign when codes have been written and some problems have been found then
    - ◊ Indispensability of UML class diagram
    - ◊ General purpose of XMI
    - ◊ Powerful reasoning ability of Situation Calculus

    Integrated by UCVSC
- There is still much work for further research:
  - The final step is not automatic but manual
  - Most elements in UML class diagram are referred to but not entirely
  - Other UML diagrams and more formal languages should be involved
  - More intelligent factors such as accessibility of verification options configurable by users would be concerned (aim of our new platform)

References

[1] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Pearson Higher Education, Boston, 2005.

[2] OMG: UML 2.0 Superstructure Specification, version ptc/04-10-02. Object Management Group, Inc., Needham, MA, http://www.omg.org, 2004.

[3] M. Gogolla, F Büttner. M. Richters, "USE: A UML-based specification environment for validating UML and OCL," Science of Computer Programming, vol. 69, pp. 27-34, 2007.

[4] R.V.D. Straeten, T. Mens, J. Simmonds, V. Jonckers, "Using description logic to maintain consistency between UML models," LNCS, vol. 2863, pp. 326-340, Springer, Heidelberg, 2003.

[5] J. McCarthy, "Situations, actions and causal laws," Stanford Artificial Intelligence Project, Memo 2, 1963.

[6] M. Gogolla, P. Ziemann, S. Kuske, "Towards an Integrated Graph Based Semantics for UML," Electr. Notes Theor. Comput. Sci, vol. 72, 2003.

[7] J. Kong, K. Zhang, J. Dong, D. Xu, "Specifying Behavioral Semantics of UML Diagrams through Graph Transformations," The Journal of Systems and Software, vol. 82, pp. 292-306, 2009.

[8] J. Dong, Y. Zhao, Y. Sun, "XSLT-based evolutions and analyses of design patterns". Software: Practice and Experience. vol. 39, pp. 773-805, 2009.

[9] A. Knapp, S. Merz, Hugo/RT, http://www.pst.ifi.lmu.de/projekte/hugo/, 2004.

[10] OMG, XML Metadata Interchange, version 1.2, Object Management Group, Inc., Needham, MA, http://www.omg.org/, 2002.

[11] OMG, Meta Object Facility Specification, version 1.4, Object Management Group, Inc., Needham, MA, http://www.omg.org/, 2002.

[12] B. Li, J. Iijima, "A Survey on Application of situation calculus in Business Information Systems," Proc. International Conference on Convergence Information Technology (ICCIT 07), pp. 425-431, 2007.

Thanks