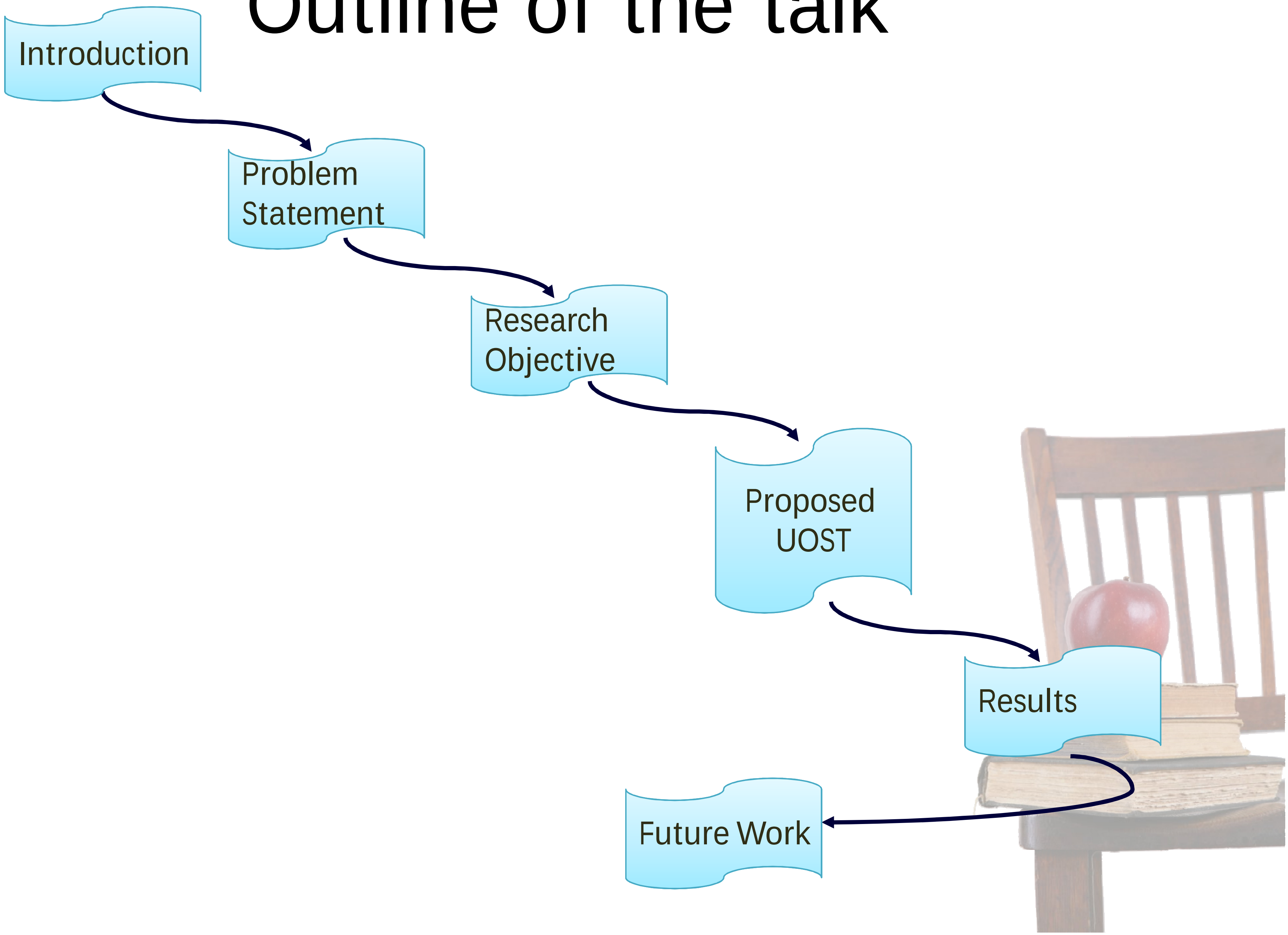
UOST: UML/OCL Aggressive Slicing Technique For Efficient Verification of Models

SAM 2010

Asadullah Shaikh
University of Southern Denmark
ashaikh@mmmi.sdu.dk

Syddansk Universitet

Outline of the talk



Introduction

Problem Statement

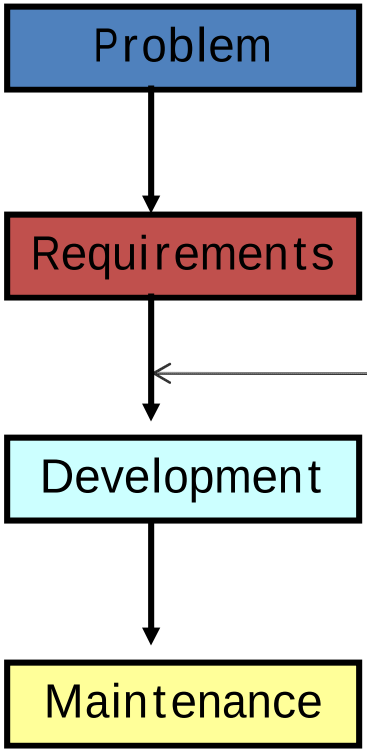
Research Objective

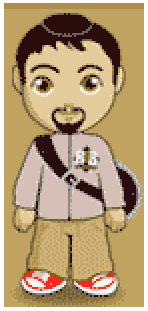
Proposed UOST

Results

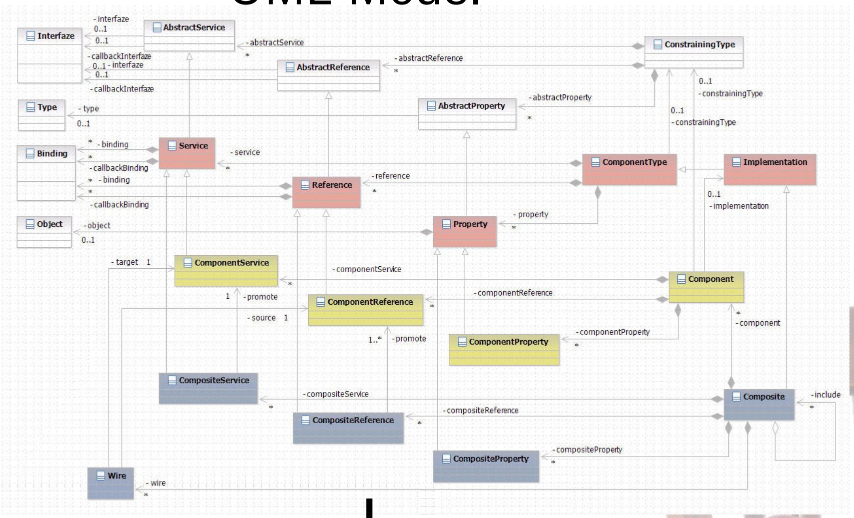
Future Work

Software Development Life Cycle





UML Model

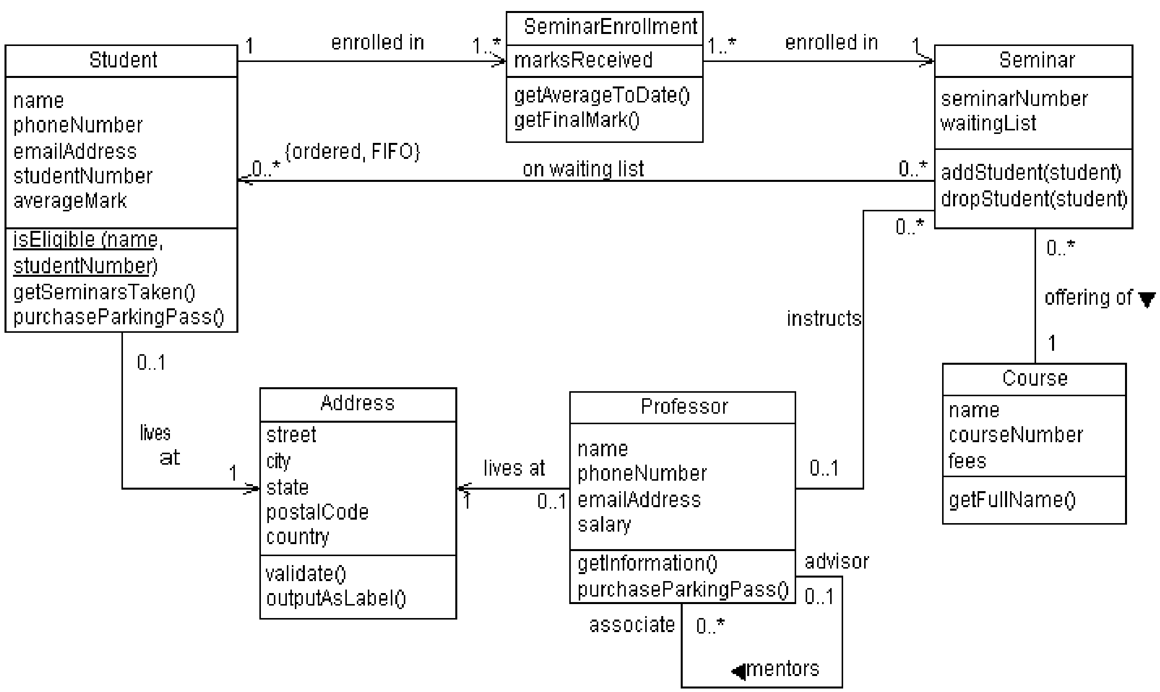



```
  --></script>
<script src="https://ssl.google-analytics.com/ga.js"
type="text/javascript"></script>
<script type="text/javascript">
var page = typeof urchinPage != 'undefined' ? urchinPage
:
window.location.href;
var account = 'UA-18065-1';
if (typeof _gat != 'undefined') {
var pageTracker = _gat._getTracker(account);
pageTracker._setAllowAnchor(true);
pageTracker._initData();
pageTracker._trackPageview(page);
}
</script>
<script type="text/javascript"><!--
      _uacct = "UA-84013-5";
      _uff = 0;
      urchinTracker('Google_Calendar_Mobile_Sync');
    --></script>
</body>
</html>
```

Brief Definitions

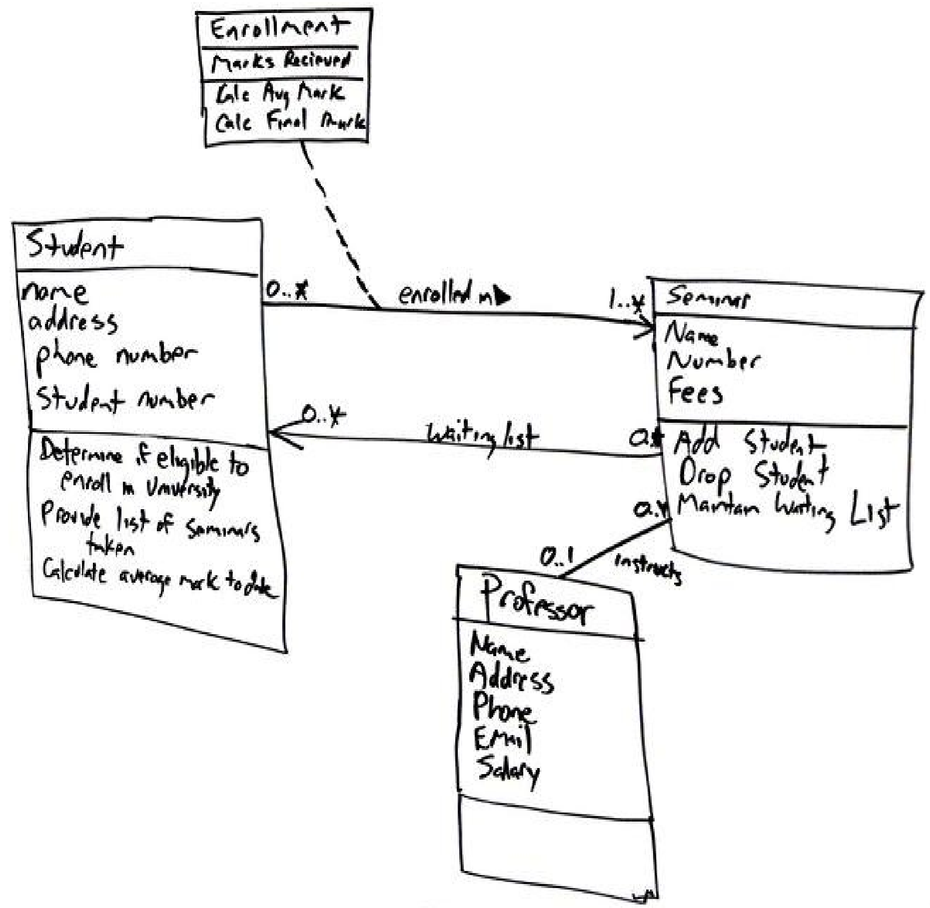
- UML--includes a set of graphical notation techniques to create abstract models of specific systems, referred to as UML model.

- Class diagram describes the structure of the system.



Brief Definitions

- Class Diagram
  - Attributes
  - Operations
- It consists several relationships
  - Association
  - Aggregation
  - Composition
  - Generalization etc

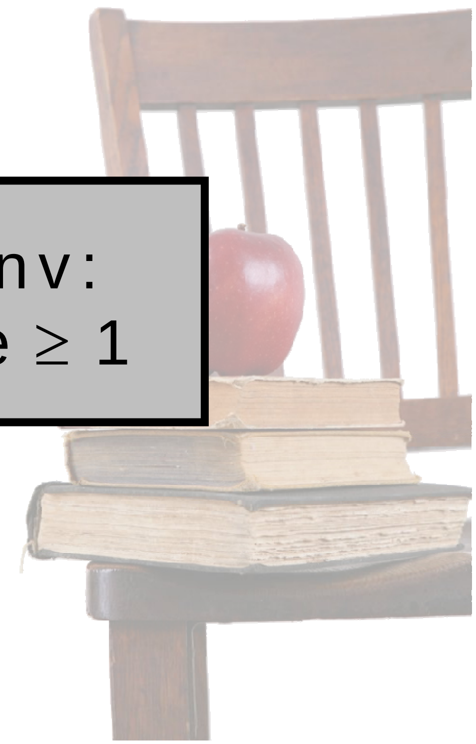


Brief Definitions

- OCL-- declarative language for describing rules upon UML models notation to express the integrity constraints which cannot be captured by the graphical UML notation.

Class
Id: string studentSize:int

```
context Class inv:
self.studentSize ≥ 1
```



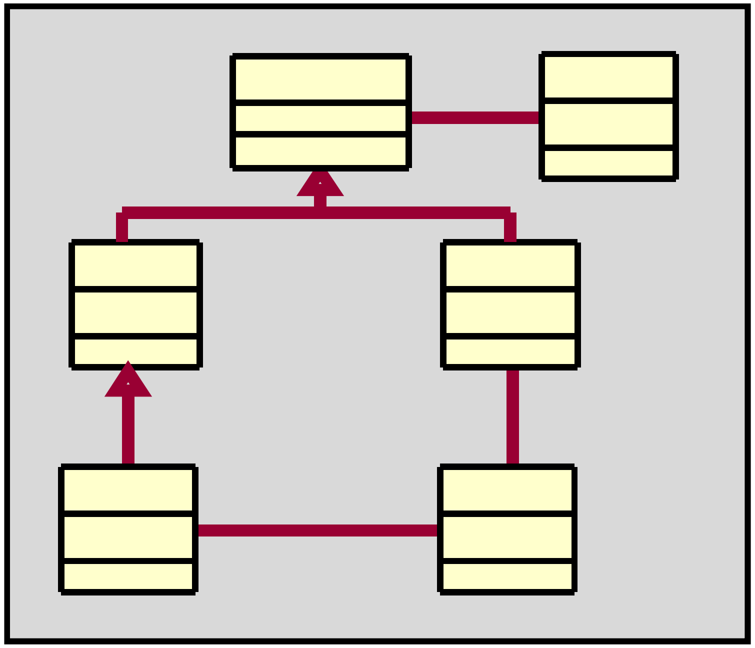
Brief Definitions

- Verification of Model--ensures that the model is correct and does not contain any kind of errors. “Are we building the model right”.

- Validation of Model-- ensures that the model accomplish all the requirement. “Are we building the right model”.

“We will focus on Verification Process of UML/OCL Model”

How UML/OCL Model Look Like...



```
context Person inv:
self.age ≥ 0
```

Graphical constraints:

Multiplicities of associations

Inheritance hierarchies

Textual constraints:

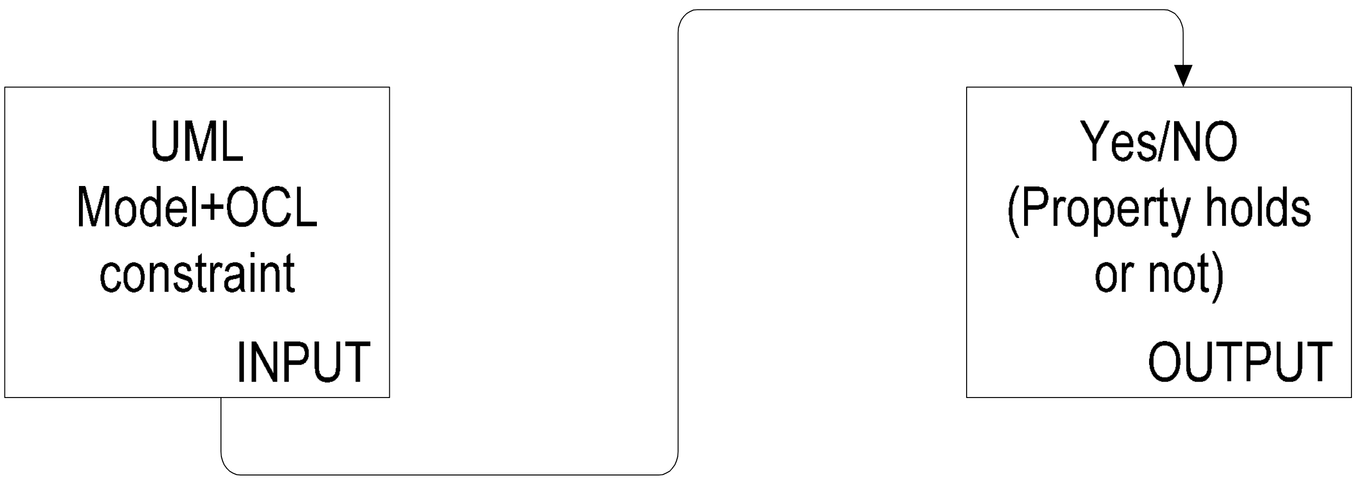
Complex class invariants

Model Correctness (Verification)

Class	Satisfying Object	Unsatisfying Object
Person name: string age:int	**P1:Person** name = ‘Tony’ age = 23	**P2:Person** name = ‘Tom’ age = -3

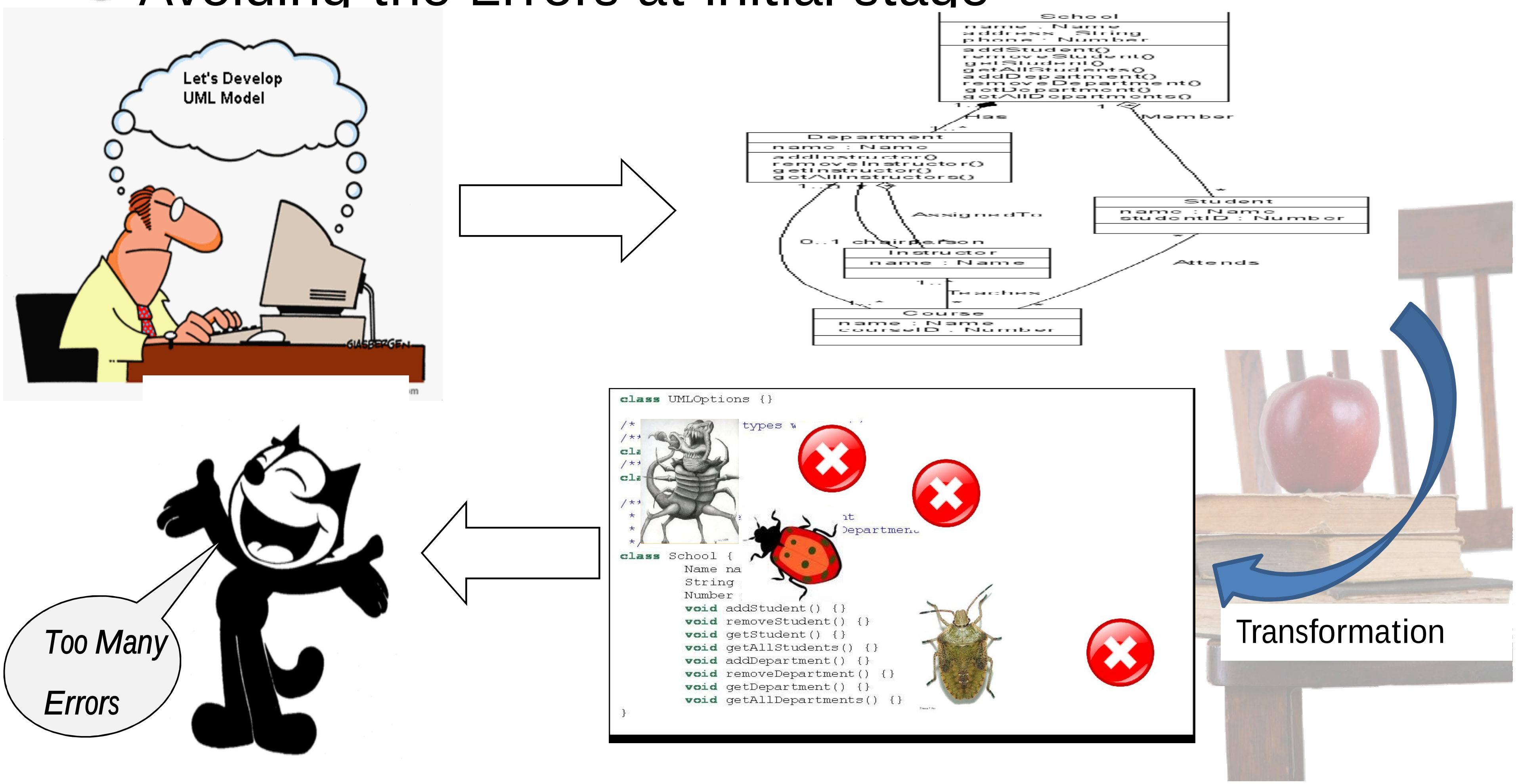
```
// Person’s age must be more then zero
context Person inv: self.age>0
```

INPUT & OUTPUT



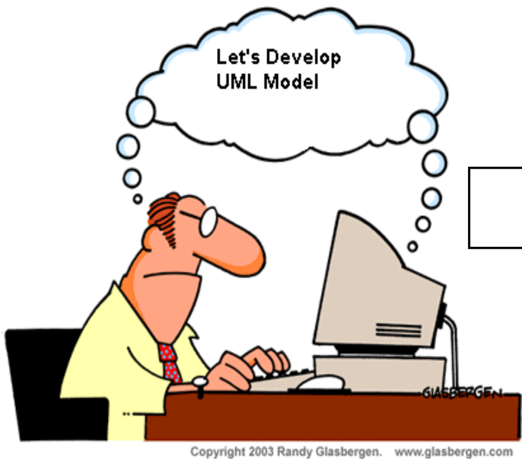
WHY Verification & Validation

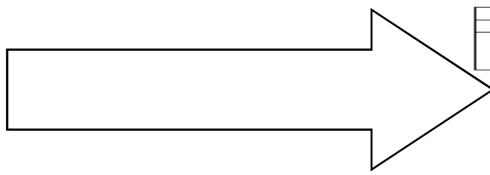
- Error in the model means Error in the code
- Avoiding the Errors at initial stage

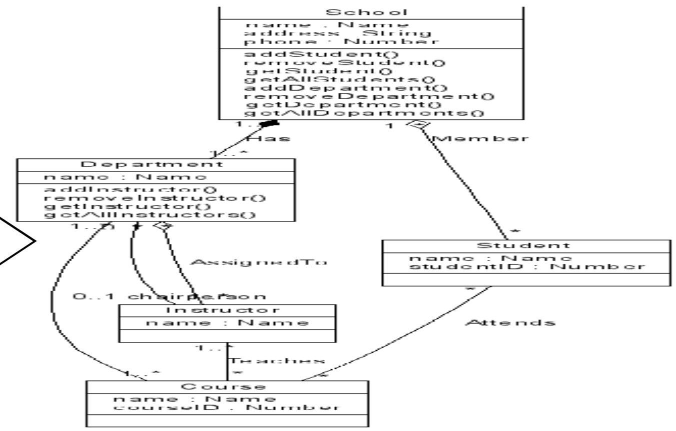


WHY Verification & Validation

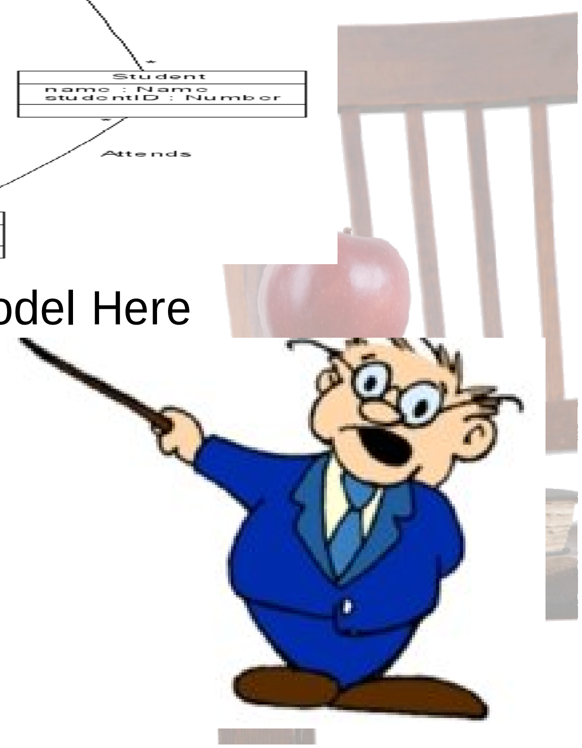
- Error in the model means Error in the code
- Avoiding the Errors at initial stage



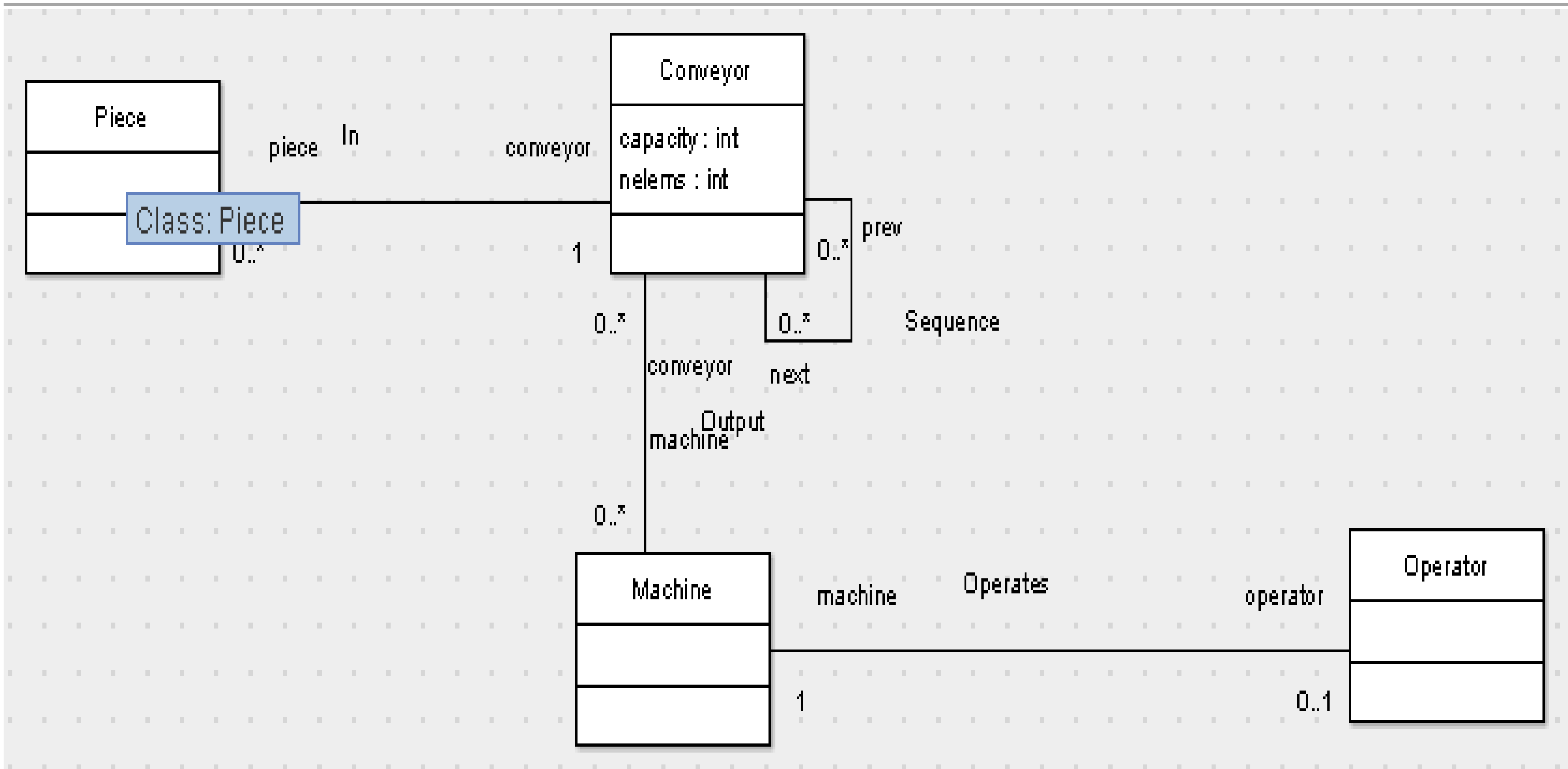




It is Better If I Verify My Model Here

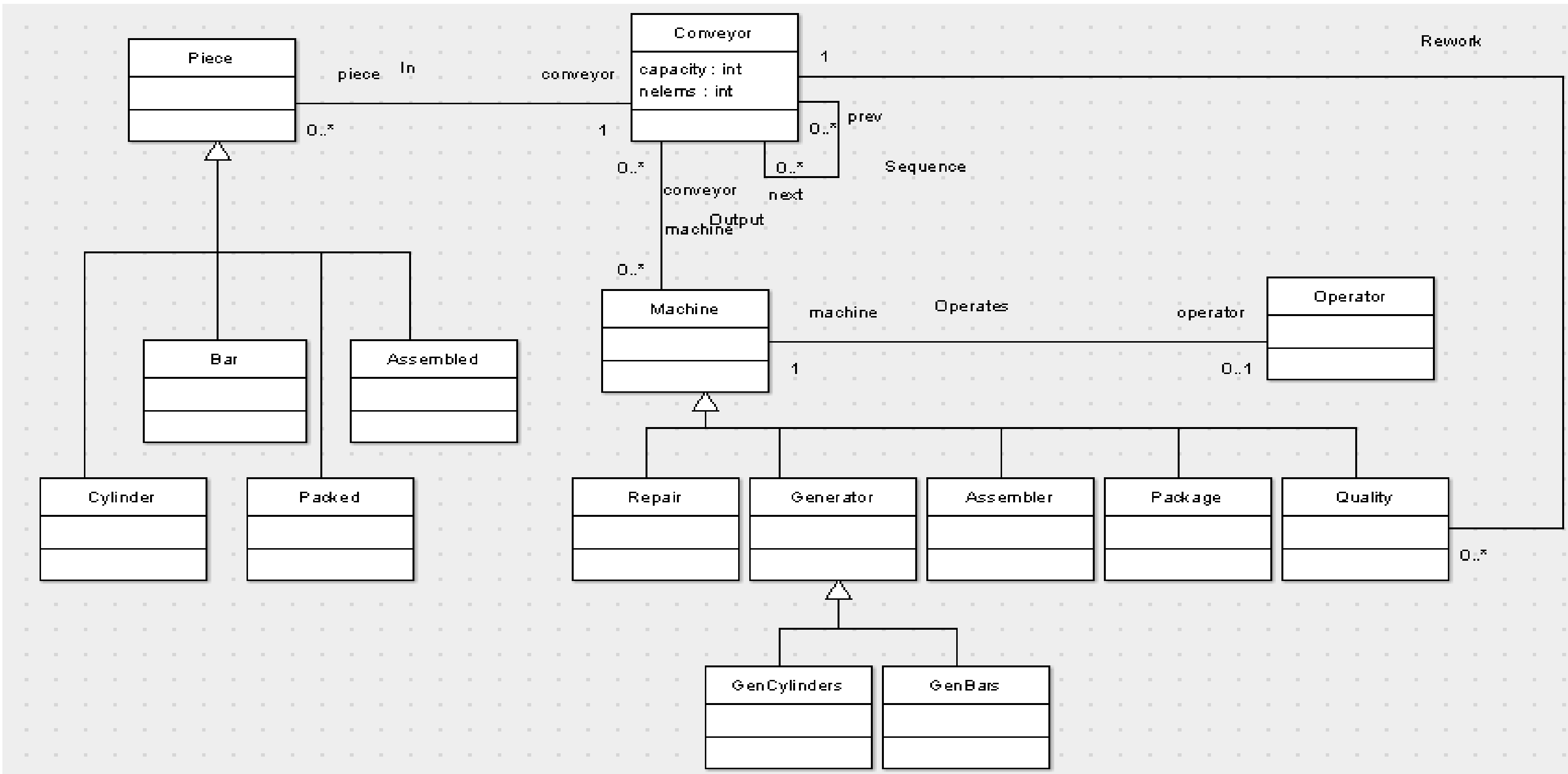


Example (Estimated time)

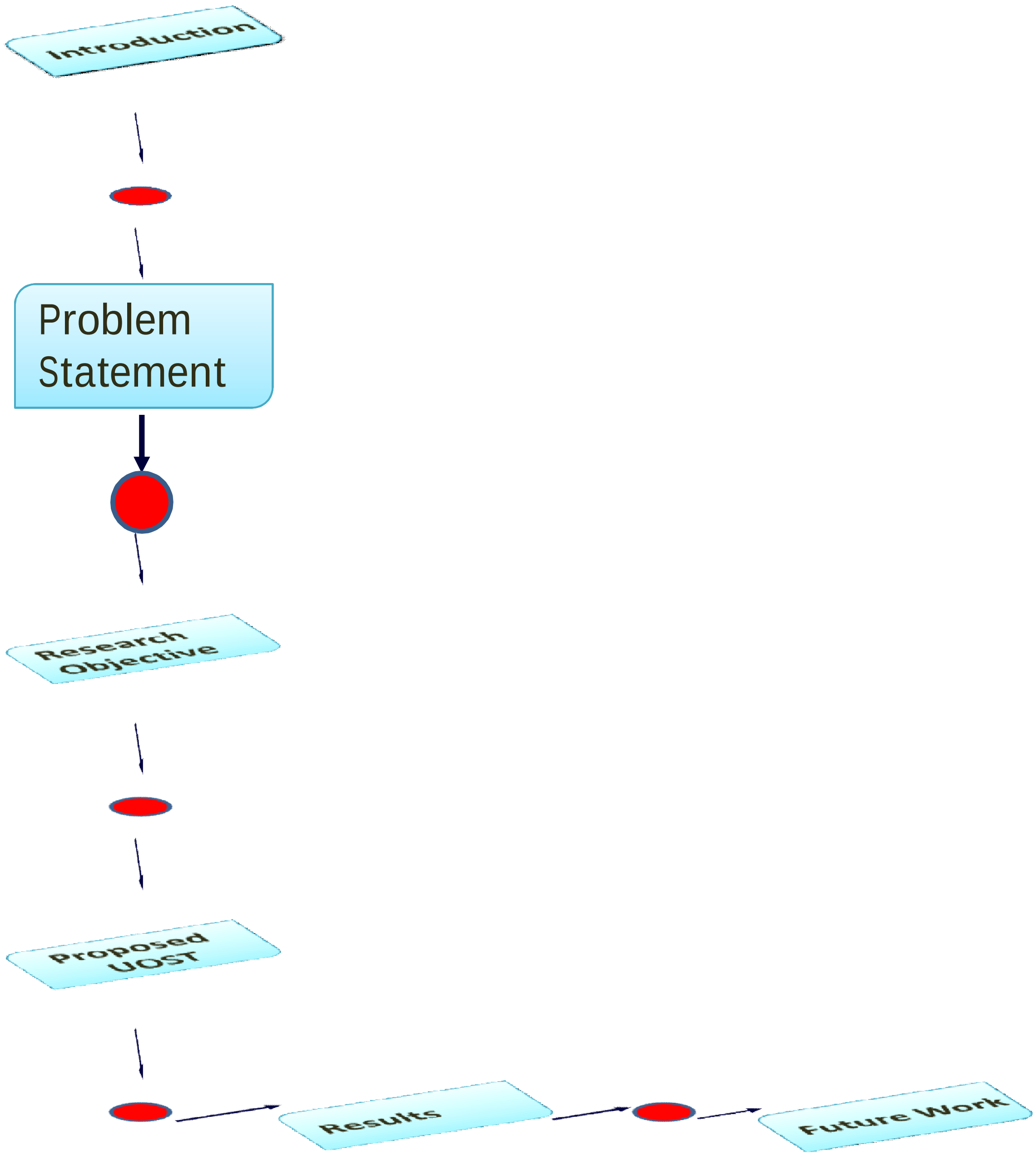


- UMLtoCSP takes 92 second to verify the property.

Example (Estimated time)



- New classes that do not affect the property being verified.
- 2 hours (did not verified any property)


Introduction
Problem Statement
Research Objective
Proposed UOST
Results
Future Work

Identifying the Problem

- Verification of small models is supported by most of tools and methods .

So where is the problem?

Identifying the Problem

- Verification of large and complex UML/OCL model.
- Efficient and scalable method.
- At certain point, available verification methods stop working.
  - CPU time
  - Memory
  - Allocated resources

Identifying V&V Tools

- V&V Tools Available in a market.
  - UMLtoCSP
  - HOL-OCL
  - MOVA
  - ALLOY
  - UML2ALLOY
  - USE
  - CQC (Is method)
  - Other methods (not implemented)

Proposed Benchmark

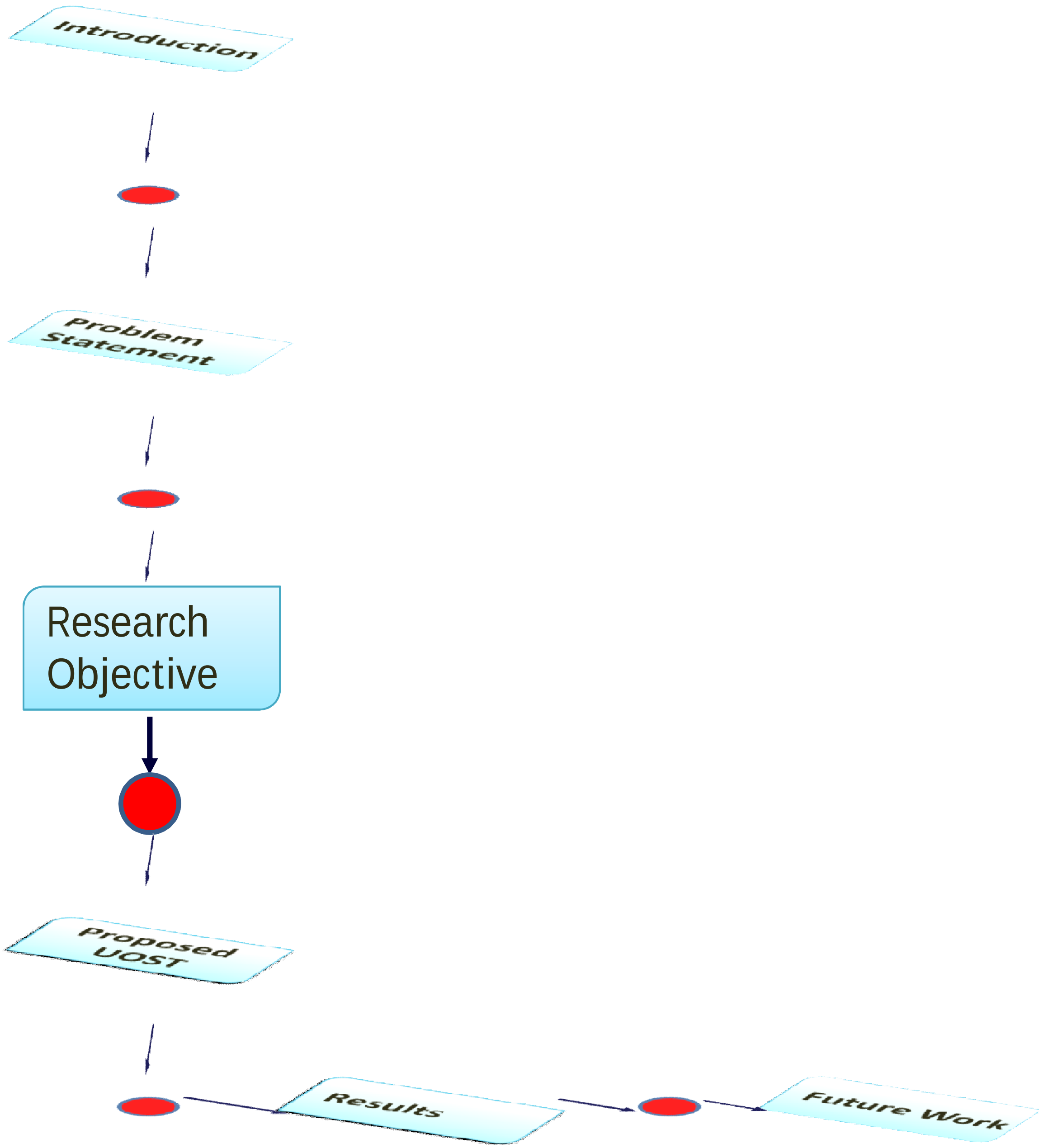
Example	Classes	Associations	Attributes	Invariants	Verification Time
Atom-Molecule	2	1	6	1	0.03s
Paper-Researcher	2	2	5	4	0.04s
Coach	15	10	27	6	5008.76s
Production System	50	30	72	5	3605.35s
Company	100	100	100	100	6233.35s
Script 1	100	53	122	2	Time-out
Script 2	500	227	522	5	Time-out
Script 3	1000	505	1022	5	Time-out
Cycle 1	10	10	10	10	Time-out
Cycle 2	100	100	100	100	Time-out

Table 1 Description of the UML/OCL benchmarks and Verification Time.

Scope	TT	ST	TT+ST = TVT
2	125ms	47ms.	172ms
3	187ms	78ms	265ms
4	281ms	172ms	453ms
5	473ms	190ms	663ms
6	671ms	344ms	1015ms
7	969ms	484ms	1453ms
....	....	....	...
10	Time-out	Time-out	Time-out

TT Translation Time **ST** Solving Time
TVT Total Verification Time

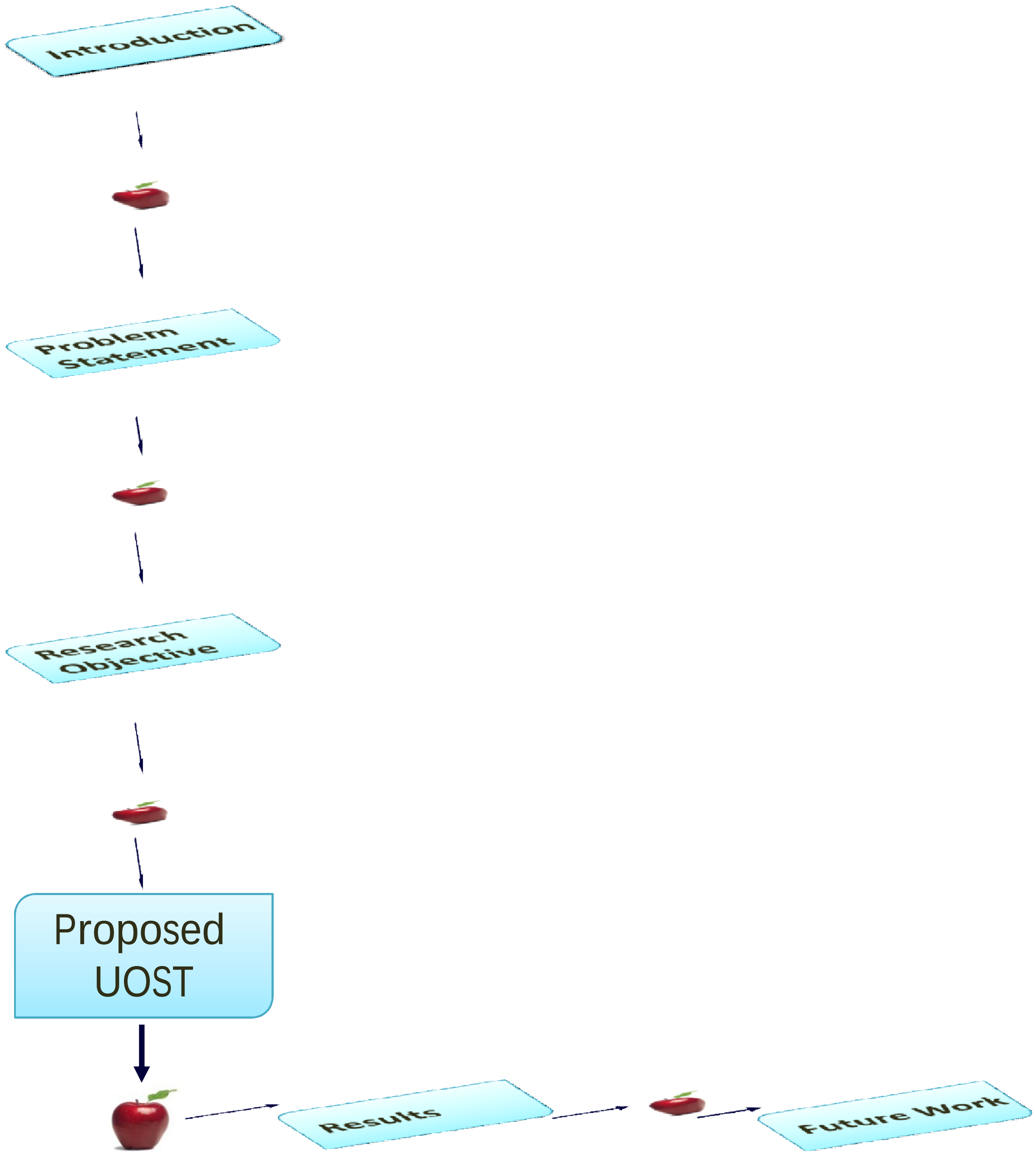
Table 2 Description of experimental results (Alloy).


Introduction
Problem Statement
Research Objective
Proposed UOST
Results
Future Work

Research Objective

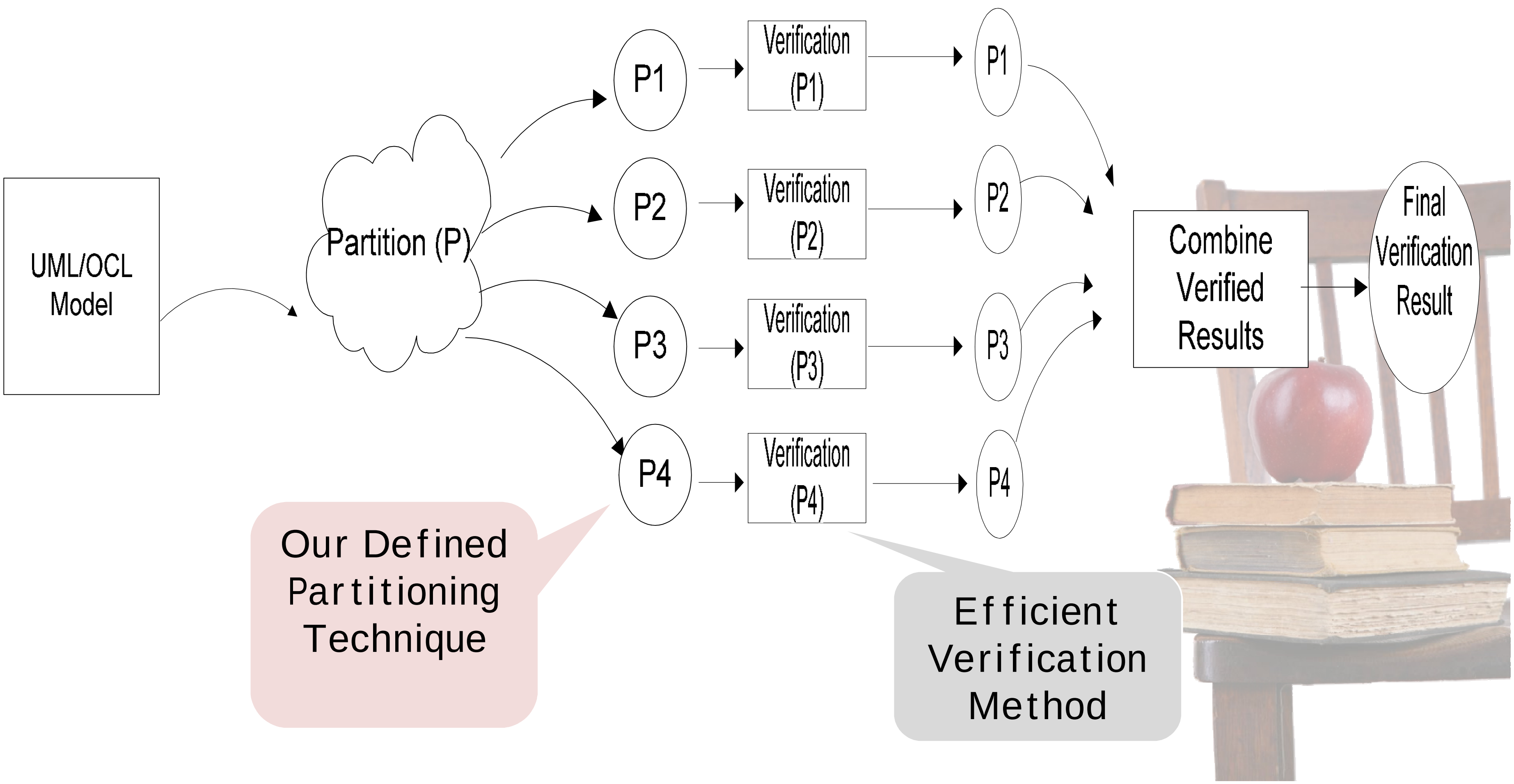
- To design a Framework/Technique for improving the scalability for verification process.

- Efficient verification technique usable to large models.
  - Abstract irrelevant components of the system.
  - Partition model into independent submodels.


Introduction
Problem Statement
Research Objective
Proposed UOST
Results
Future Work

UML/OCL Slicing Technique

- Slicing a model into small fragments (submodels).



UML/OCL Slicing Technique

- Our goal is to determine whether the input class diagram has legal instances.

- Two different notions of satisfiability is considered:
  - Strong and Weak satisfiability.

UML/OCL Slicing Technique

- Given a Model `m', into m1, m2, m3…mn submodels, where m is satisfiable if all m1, m2, m3…mn submodels are satisfiable.

- Each submodel is a subset of the OCL constraints.

UML/OCL Slicing Technique

- Each slice is less constrained than the original model.
- It is necessary to ensure that if the original model is unsatisfiable.

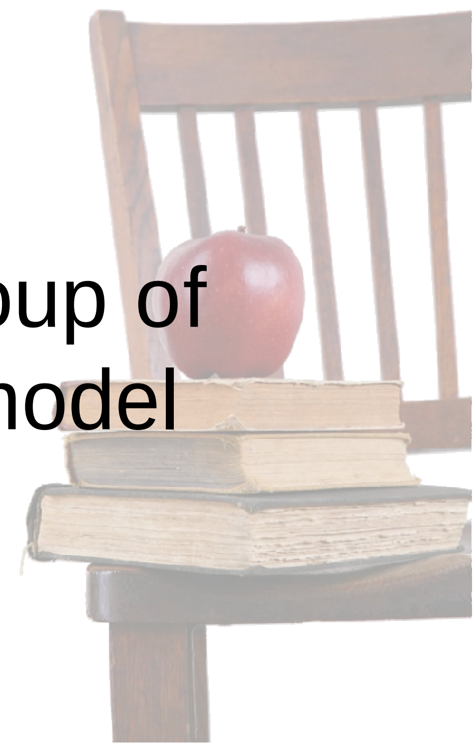
Algorithm 2 Slicing Algorithm

Input: Property being verified
Output: A partition P of the model M into non-necessarily disjoint submodels

```
G ← BuildFlowGraph(M) {Creating the flowgraph}
{Cluster the OCL constraints}
for each pair of constraints c1, c2 in M do
   if ConstraintSupport(M, c1) ∩ ConstraintSupport(M, c2) ≠ ∅ then
      MergeInSameCluster(c1, c2)
   end if
end for
{Work on each cluster of constraints separately}
for each cluster of constraints Cl do
   subModel ← empty model {Initialize the subModel to be empty}
   {Initialize worklist}
   workList ← Union of the ConstraintSupport of all constraints in the cluster
   while workList not empty do
      node ← first(workList) {Take first element from workList and remove it}
      workList ← workList \ node
      for each subclass or superclass c of node do
         subModel ← subModel ∪{c}
         if c was not before in the partition then
            workList ← workList ∪{c}
         end if
      end for
      for each class c loosely or tightly connected to node do
         if Property = weak SAT and tightly coupled then
            subModel ← subModel ∪{c}
         else if Property = strong SAT and tightly coupled then
            workList ← workList ∪{c}
         end if
      end for
   end while
end for
```

Cluster of Constraints

- The constraint support is the set of classes restricted by the constraint.
- Clustering of the constraint.
- Identifying the constraint or a group of constraints restricting the same model element.



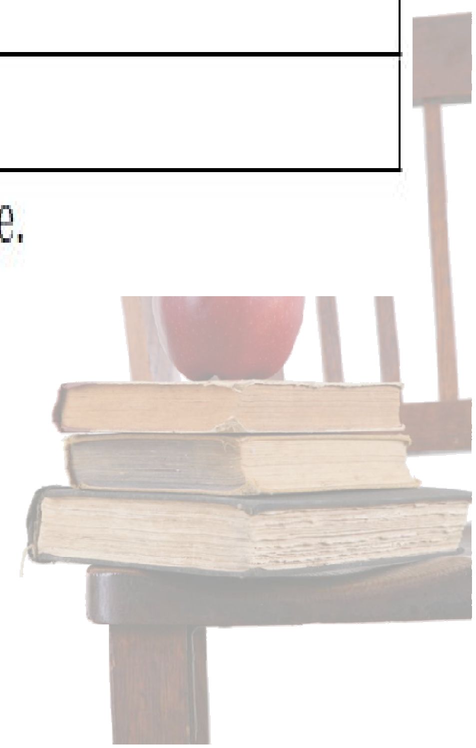
Constraint Support

- Local and Global Constraint.
- Local: If it can be evaluated by examining only the values of the attributes in one object of class C.
- Global: if the constraint are examined multiple objects of the same class.

Constraint Support

Invariant	Support	Attributes	Navigations
MaxCoachSize	Coach, Trip, Passenger	Coach.noOfSeats	tripscoach, tripspassengers
ticketNumberPositive	Ticket	Ticket.(number)	None
NonNegativeAge	Passenger	Passenger.age	None

Table 1. Support, attributes and navigations in the running example.



Flow graph Concept

- We have used a flow graph that captures the dependencies of the elements within the UML/OCL class diagram.

- Each vertex is a class and each arc is a relationship.

UML relationship	*Loosely/Tightly Coupled*	*Edge*
Association: Lower bound ≥ 1 (e.g. 1..*)	Tightly Coupled	$\longrightarrow$
Association: Lower bound $= 0$ (e.g. 0..3)	Loosely Coupled	$\dashrightarrow$
Generalization	Tightly Coupled	——
Aggregation	Tightly Coupled	——

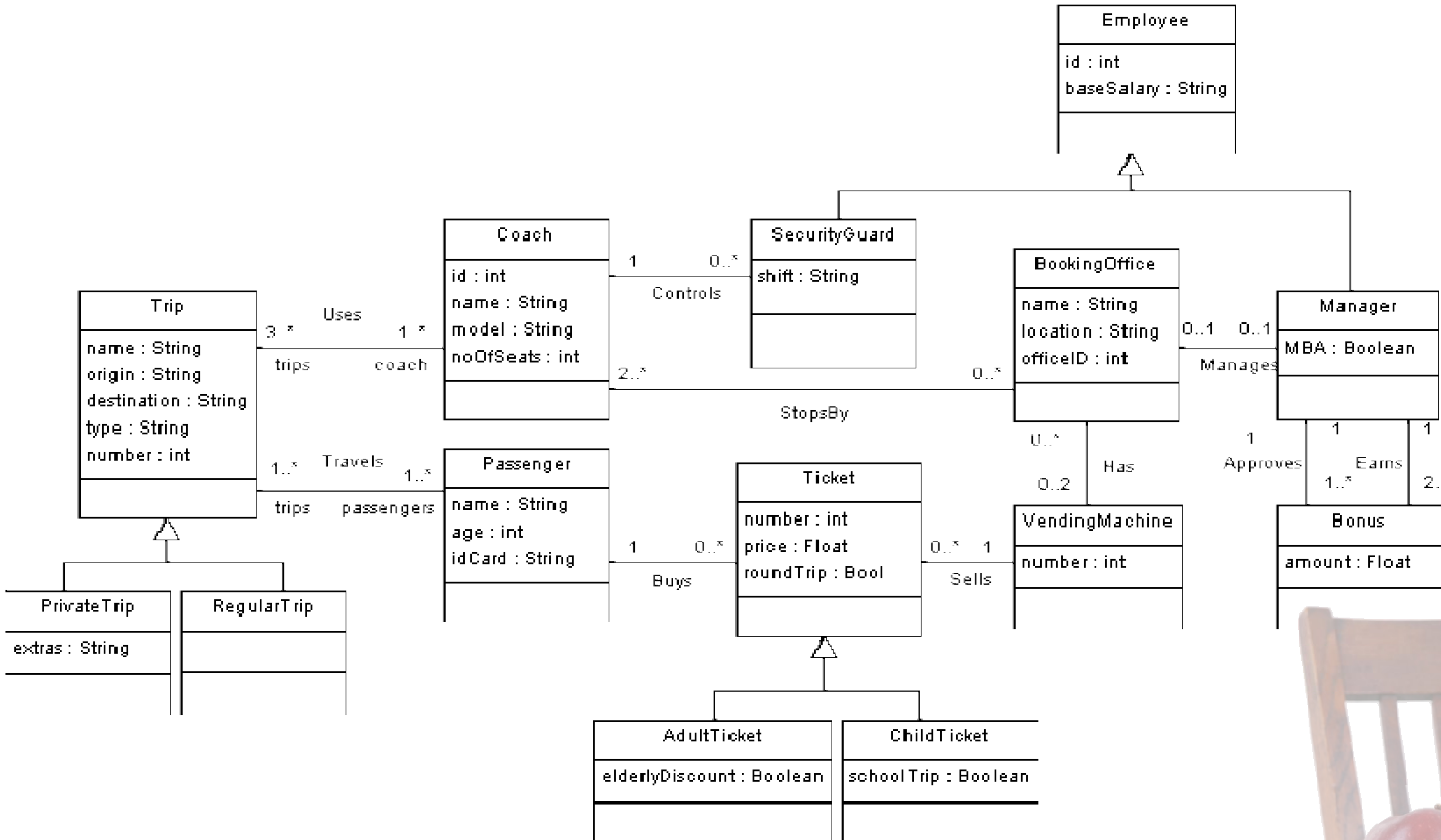
Table 2. Edge Concept of Flowgraph

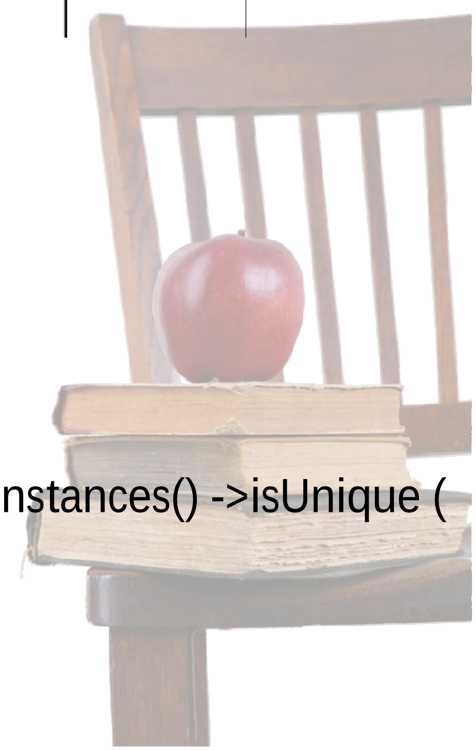
Algorithm 1 Flowgraph Creation

Input: A model M
Output: A labeled directed graph $G = \langle V, E \rangle$

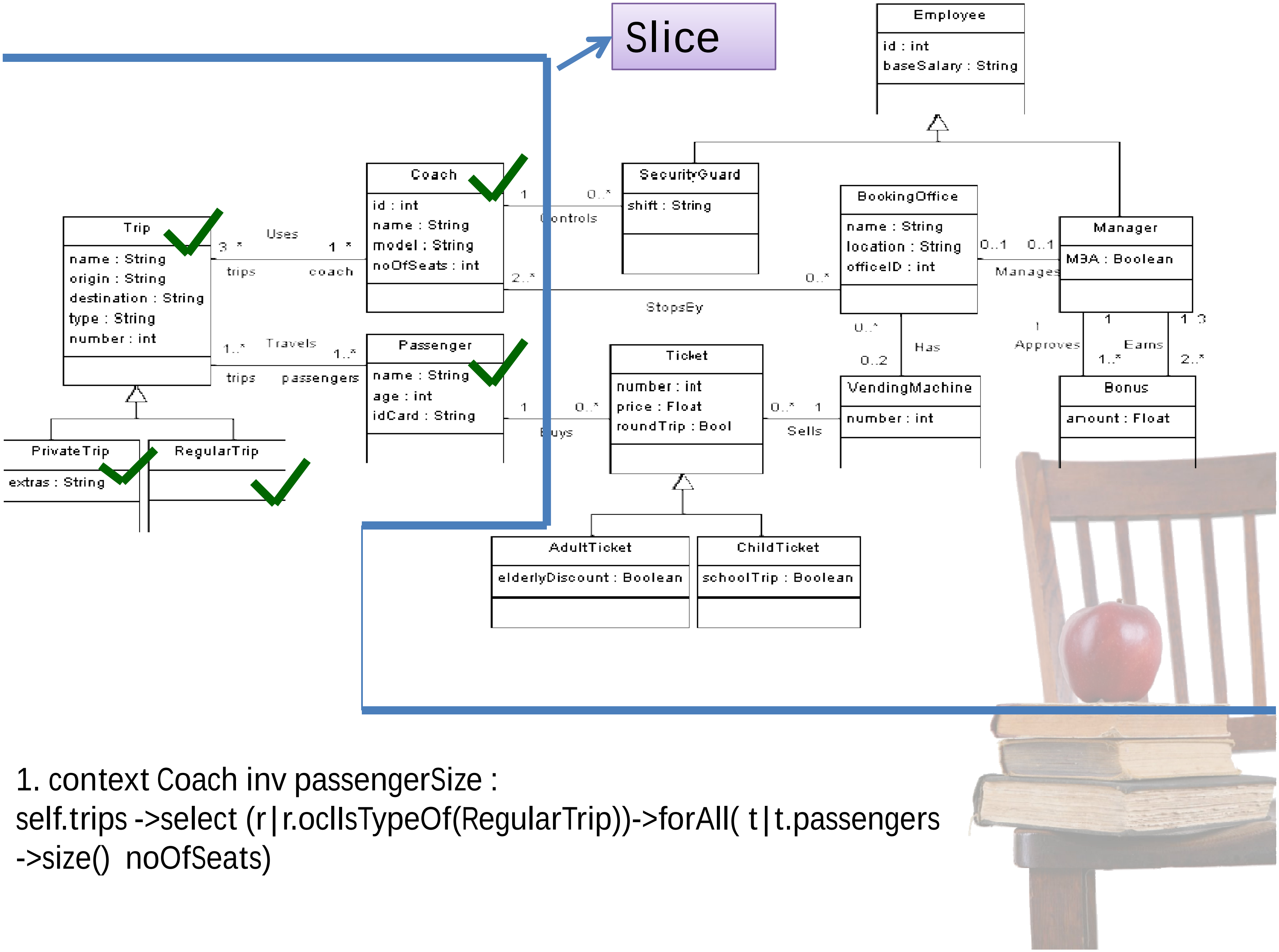
{Start with the empty graph}
Let $V \leftarrow \emptyset$ and $E \leftarrow \emptyset$
{Add all classes of the model to the flowgraph}
for class c in model M **do**
    $V \leftarrow V \cup \{c\}$
end for
{Create incoming and outgoing arcs in the flowgraph}
for each association end A in model M **do**
    $E \leftarrow (x, y)$ where x is the type of the association end and y is the type of the other class in the association
    if the lower bound of the multiplicity of A is ≥ 1 **then**
        Label the arc (x, y) as tightly associated
    else if the multiplicity A is lower bound $= 0$ **then**
        Label the arc (x, y) as loosely associated
    end if
end for
for each generalization, aggregation and composition G between classes x and y **do**
    $E \leftarrow E \cup \{(x, y)\} \cup \{(y, x)\}$
    Label the arcs (x, y) and (y, x) as tightly associated
end for

Example (Model Coach)



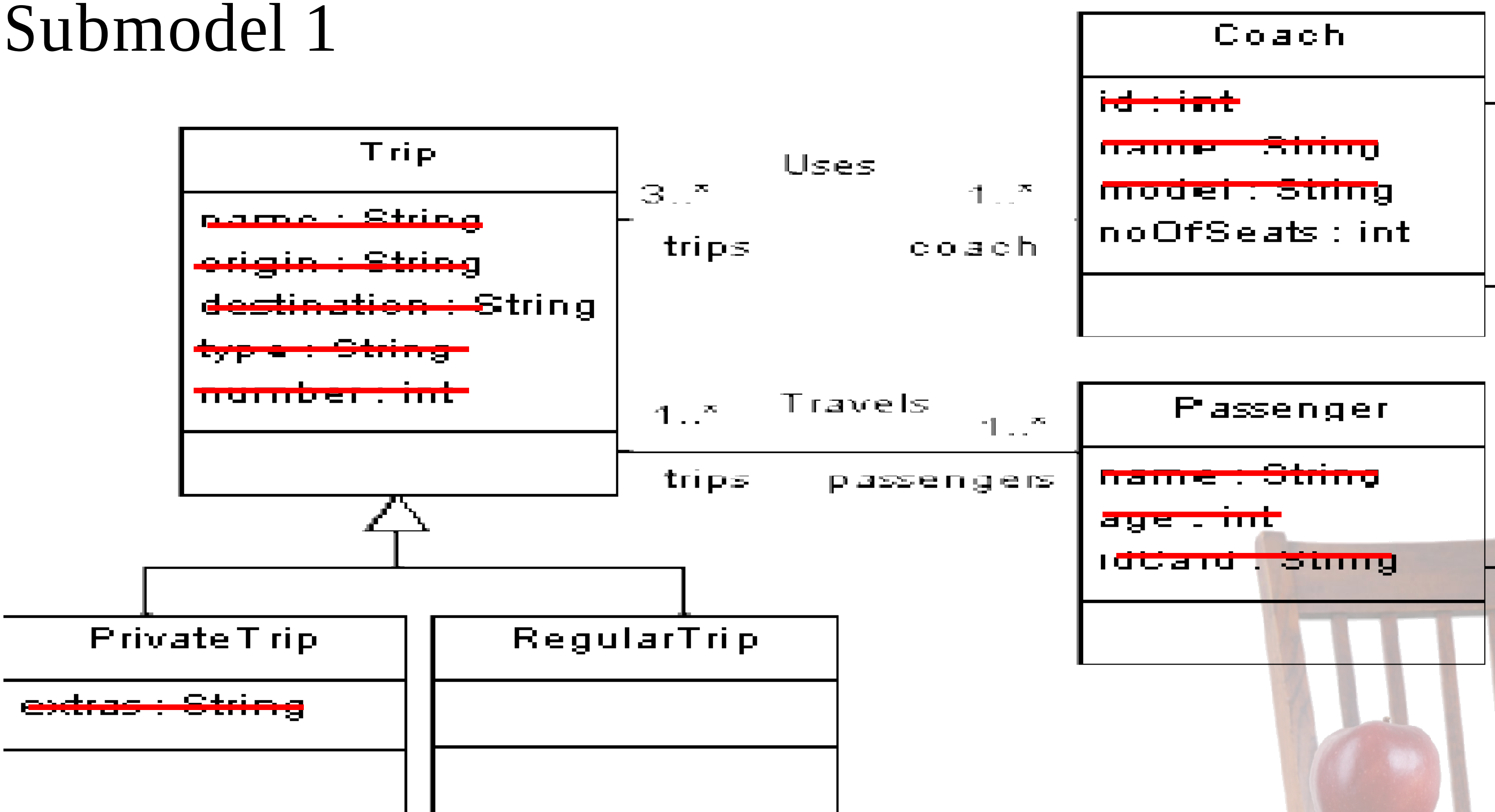


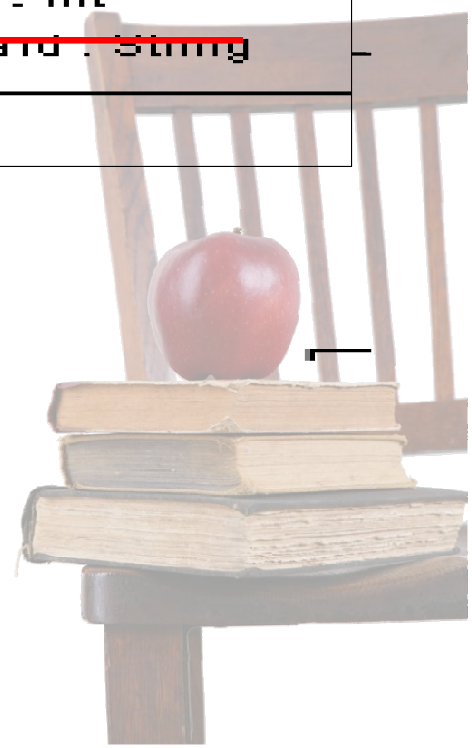
1. context Coach inv passengerSize :
self.trips ->select (r|r.oclIsTypeOf(RegularTrip))->forAll(t|t.passengers ->size() noOfSeats)
2. context VendingMachine inv UniqueNumber: VendingMachine ::allInstances() ->isUnique (t|t.number)
3. context Passenger inv NonNegativeAge:
self.age >0



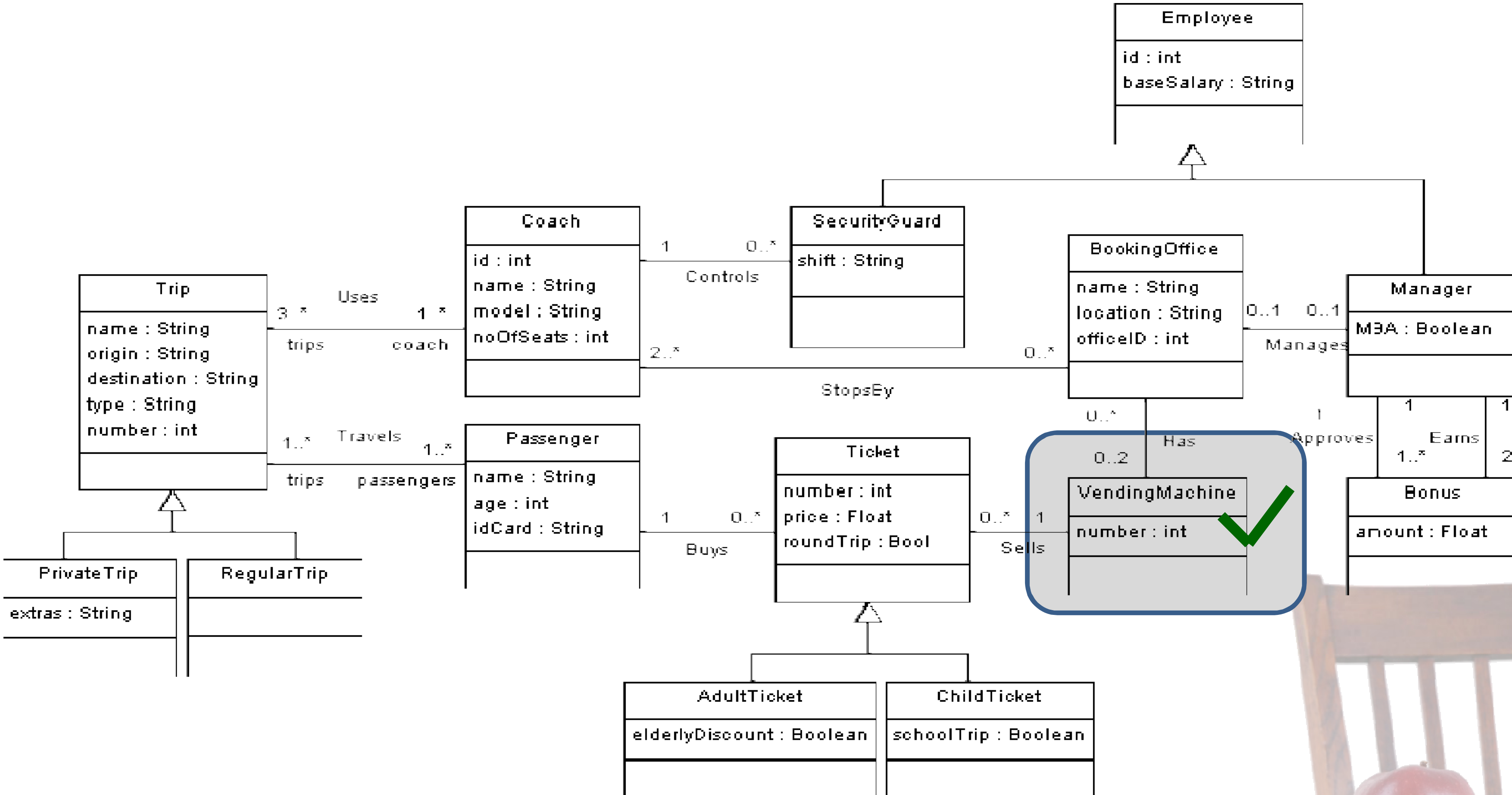
1. context Coach inv passengerSize :
self.trips ->select (r|r.oclIsTypeOf(RegularTrip))->forAll(t|t.passengers ->size() noOfSeats)

Submodel 1



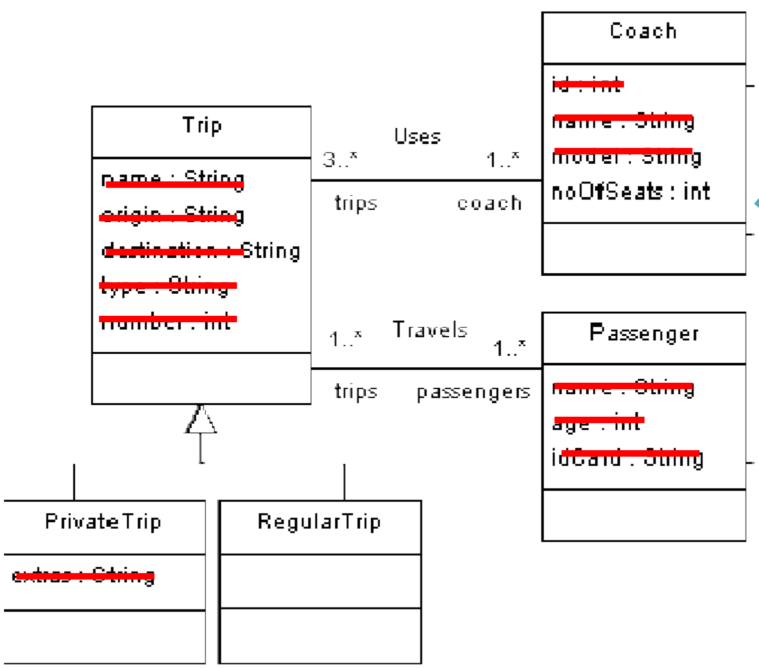


1. context Coach inv passengerSize :
self.trips ->select (r|r.oclIsTypeOf(RegularTrip))->forAll(t|t.passengers ->size() noOfSeats)



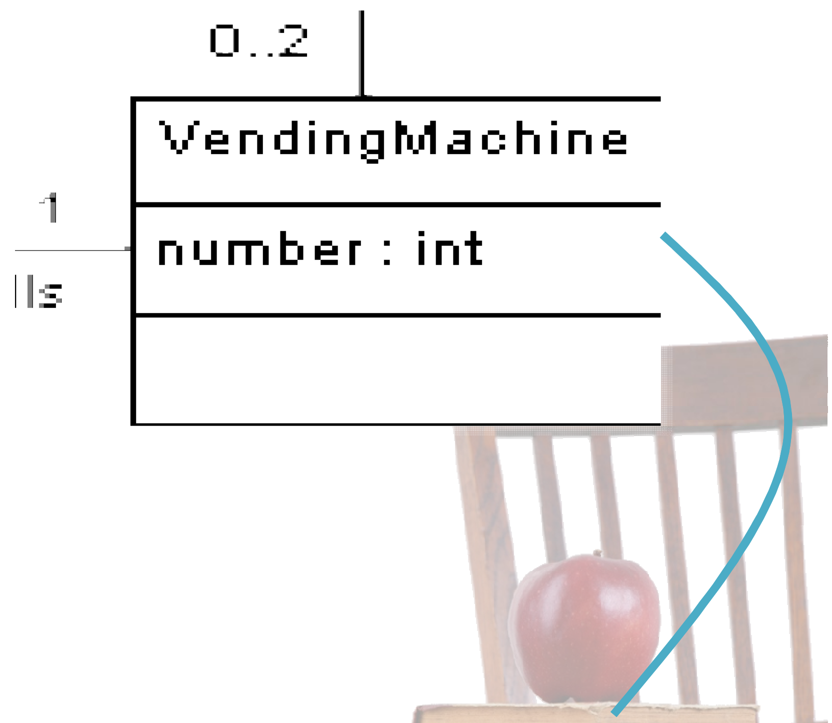
2. context VendingMachine inv UniqueNumber: VendingMachine ::allInstances() ->isUnique (t|t.number)

Submodel 1



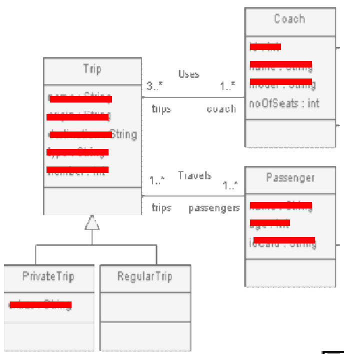
context Coach inv passengerSize :
self.trips ->select (r|r.oclIsTypeOf(RegularTrip))->forAll(t|t.passengers
->size() noOfSeats)

Submodel 2

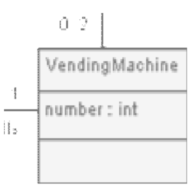


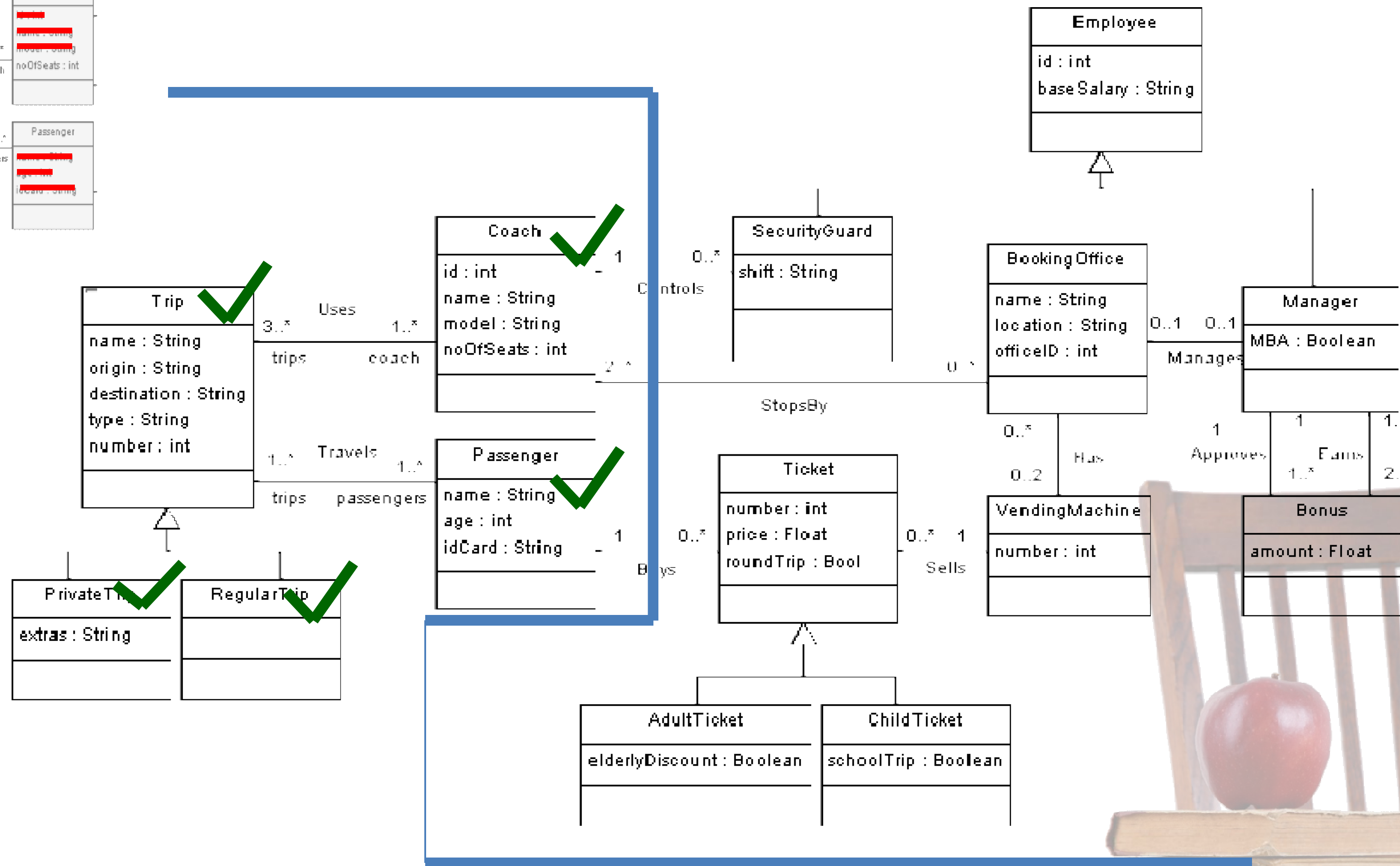
context VendingMachine inv UniqueNumber:
VendingMachine ::allInstances() ->isUnique (t|t.number)

Submodel 1



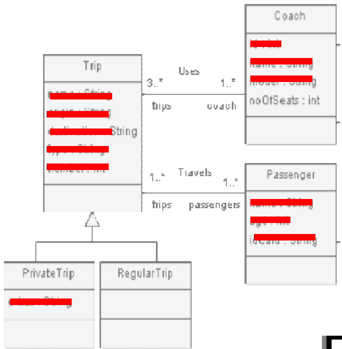
Submodel 2



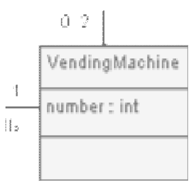


3. context Passenger inv NonNegativeAge:
self.age > 0

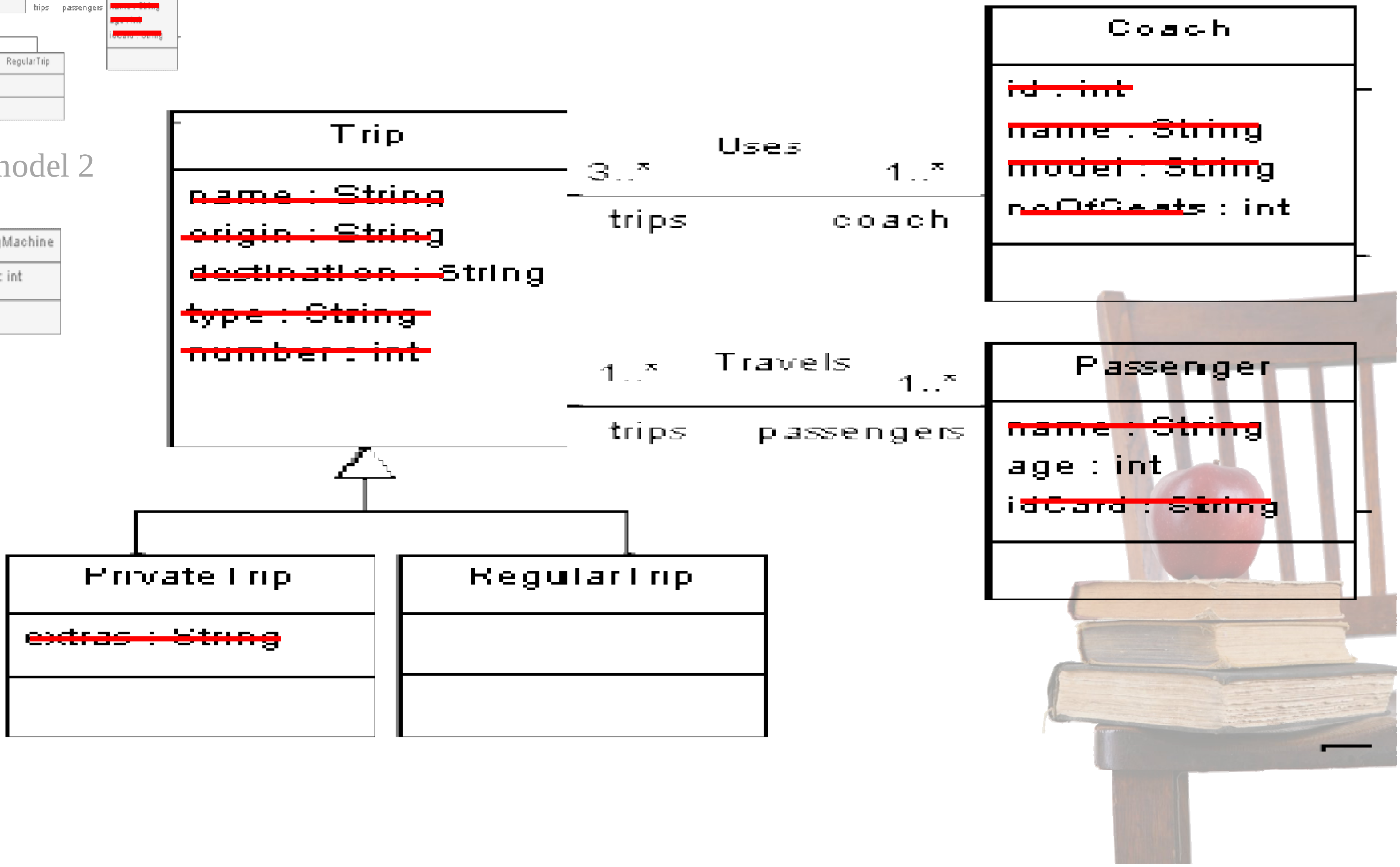
Submodel 1



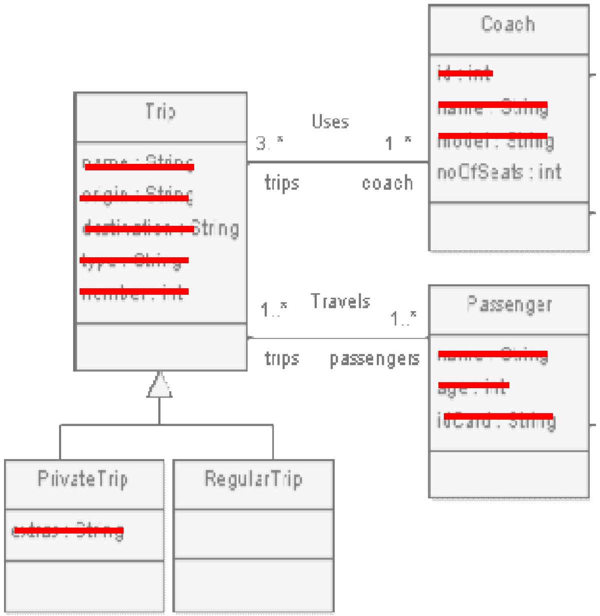
Submodel 2



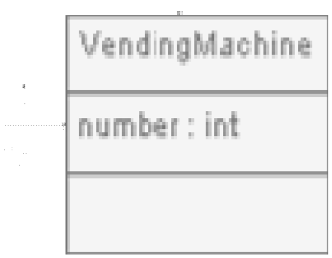
Submodel 3



Submodel 1

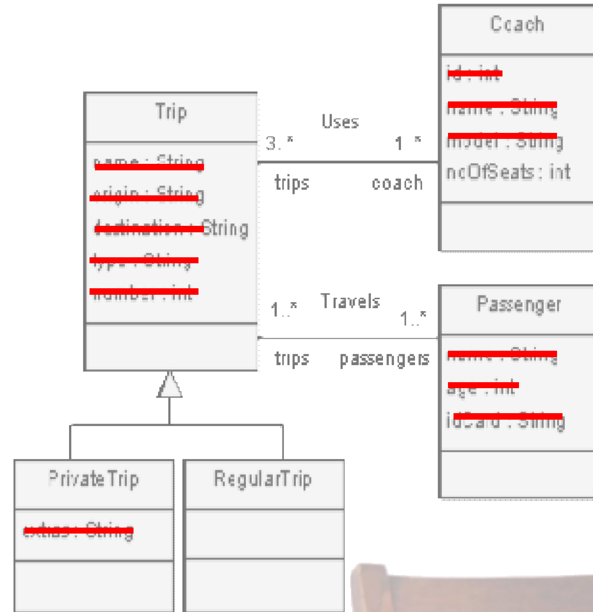


Submodel 2

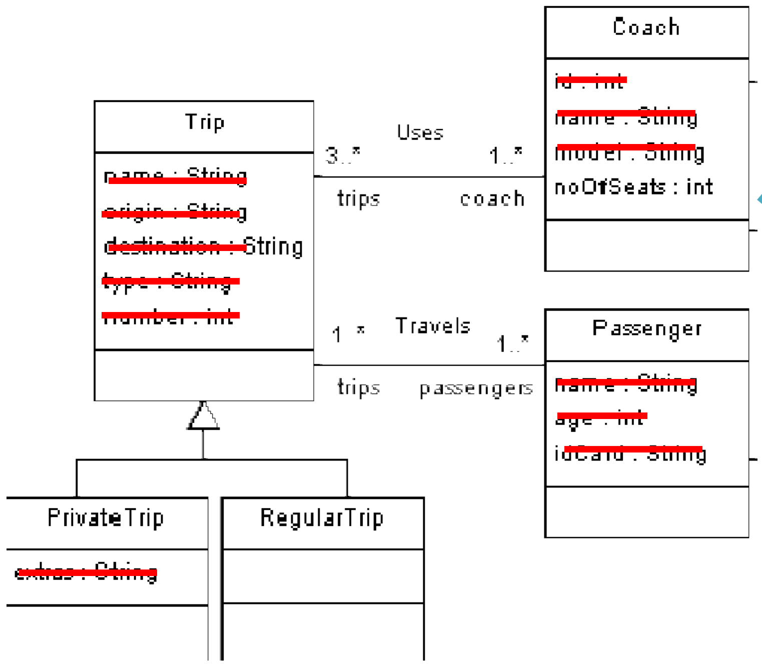


=

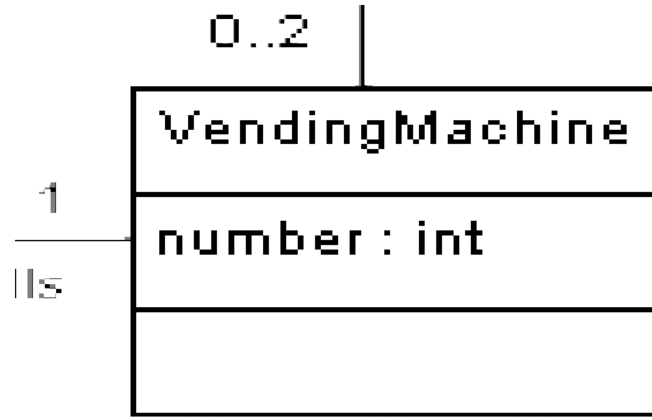
Submodel 3



Submodel 1 and 3

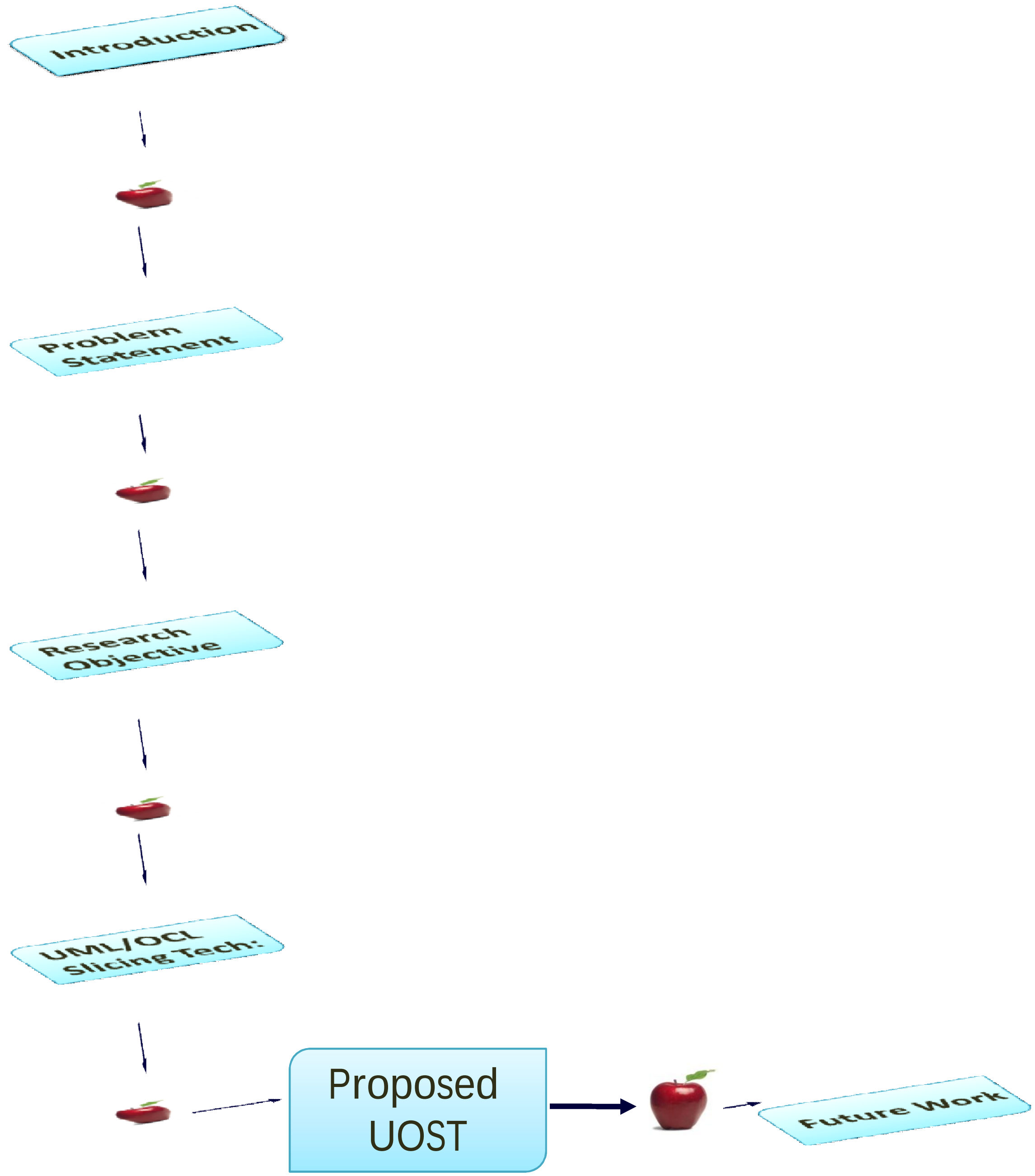


Submodel 2



1. context Coach inv passengerSize :
self.trips ->select (r|r.oclIsTypeOf(RegularTrip))->forAll(t|t.passengers
->size() noOfSeats)
3. context Passenger inv NonNegativeAge:
self.age > 0

2. context VendingMachine inv UniqueNumber:
VendingMachine ::allInstances() ->isUnique (t|t.number)


Introduction
Problem Statement
Research Objective
UML/OCL Slicing Tech:
Proposed UOST
Future Work

Example	Classes	Associations	Attributes	Invariants
Paper-Researcher	2	2	6	1
Coach	15	12	2	2
Tracking System	50	60	72	5
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 3. Description of the Examples.

Before Slicing (UMLtoCSP)			After Slicing (UMLtoCSP UOST)				
Classess	Attributes	OVT	Attributes	ST	SVT	TVT	Speedup %
2	6	2506.55s	3	0.00s	0.421s	0.421s	99.98%
15	2	5008.76s	0	0.00s	0.178s	0.178s	99.99%
50	72	3605.35s	55	0.016s	0.031s	0.047s	99.99%
100	122	Time out	117	0.016s	0.032s	0.048s	99.99%
500	522	Time out	502	0.062s	0.028s	0.090s	99.99%
1000	1022	Time out	1012	0.282s	0.339s	0.621s	99.98%

OVT Original Verification Time **ST** Slicing Time
SVT Sliced Verification Time **TVT** Total Verification Time

Table 4. Description of experimental results (UMLtoCSP).

Example	Classes	Associations	Attributes	Invariants
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	3ms	9ms	12ms	3ms	5ms	8ms	34%
3	7ms	8ms	15ms	3ms	6ms	9ms	40%
4	12ms	8ms	20ms	4ms	6ms	10ms	50%
5	17ms	10ms	27ms	4ms	9ms	13ms	52%
6	16ms	15ms	31ms	5ms	9ms	14ms	55%
7	19ms	15ms	34ms	6ms	9ms	15ms	56%

TT Translation Time　　**ST** Solving Time
STT Sliced Translation Time　　**SST** Sliced solving Time

Table 5. Slicing results in Alloy for Atom-Molecule example.

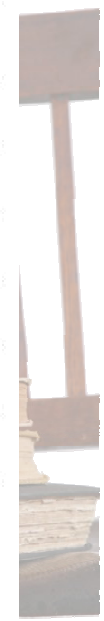
Example	Classes	Associations	Attributes	Invariants
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	7ms	10ms	17ms	3ms	5ms	8ms	53%
3	14ms	19ms	33ms	5ms	8ms	13ms	61%
4	28ms	20ms	48ms	7ms	10ms	17ms	62%
5	36ms	31ms	67ms	12ms	15ms	27ms	65%
6	45ms	50ms	95ms	17ms	15ms	32ms	67%
7	81ms	77ms	158ms	34ms	17ms	51ms	68%

TT Translation Time **ST** Solving Time
STT Sliced Translation Time **SST** Sliced solving Time

Table 6. Slicing results in Alloy for university example.



Example	Classes	Associations	Attributes	Invariants
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	20ms	46ms	66ms	5ms	8ms	13ms	81%
3	83ms	91ms	174ms	9ms	11ms	20ms	89%
4	96ms	185ms	254ms	13ms	11ms	24ms	90%
5	158ms	173ms	332ms	20ms	12ms	32ms	90%
6	233ms	367ms	600ms	25ms	23ms	48ms	92%
7	325ms	495ms	820ms	30ms	28ms	58ms	93%

TT Translation Time **ST** Solving Time
STT Sliced Translation Time **SST** Sliced solving Time

Table 7. Slicing results in Alloy for ATM machine.

Example	**Classes**	**Associations**	**Attributes**	**Invariants**
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	110ms	133ms	243ms	7ms	9ms	16ms	93%
3	161ms	290ms	451ms	9ms	9ms	18ms	96%
4	224ms	591ms	815ms	14ms	12ms	26ms	97%
5	349ms	606ms	955ms	17ms	16ms	33ms	97%
6	589ms	1077ms	1666ms	27ms	25ms	52ms	97%
7	799ms	1392ms	2191ms	38ms	25ms	63ms	97%

TT Translation Time **ST** Solving Time
STT Sliced Translation Time **SST** Sliced solving Time

Table 8. Slicing results in Alloy for script 1 (100 classes).

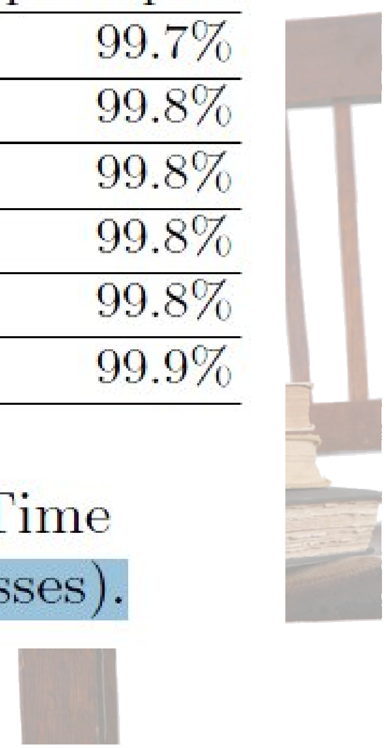
Example	Classes	Associations	Attributes	Invariants
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	1839ms	3021ms	4860ms	6ms	7ms	13ms	99.7%
3	2567ms	7489ms	10056ms	11ms	8ms	19ms	99.8%
4	3374ms	8320ms	11694ms	14ms	9ms	23ms	99.8%
5	4326ms	21837ms	26163ms	18ms	14ms	32ms	99.8%
6	5231ms	32939ms	38170ms	25ms	14ms	39ms	99.8%
7	6477ms	59704ms	66181ms	35ms	16ms	51ms	99.9%

TT Translation Time **ST** Solving Time
STT Sliced Translation Time **SST** Sliced solving Time

Table 9. Slicing results in Alloy for script 2 (500 classes).



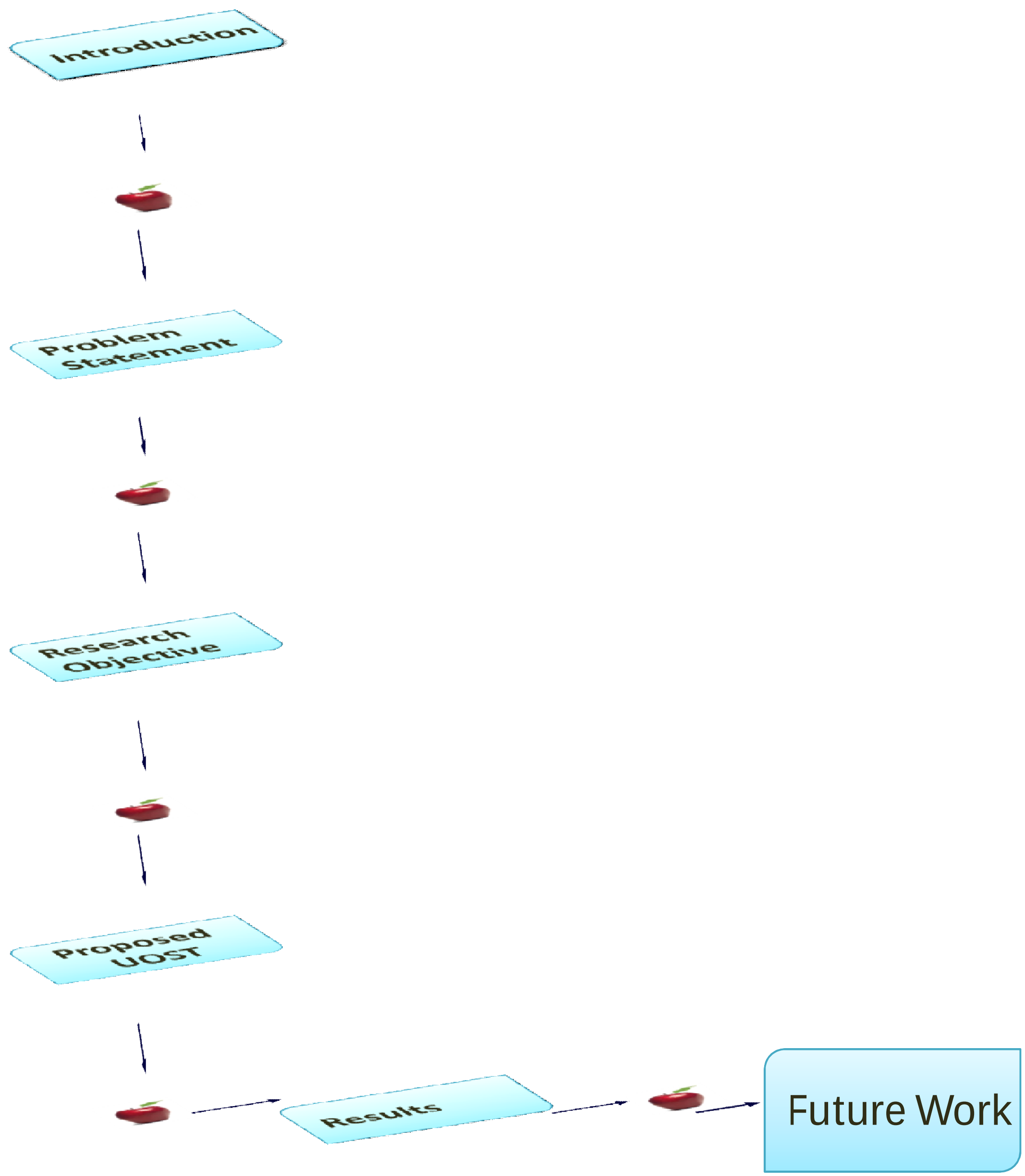
Example	Classes	Associations	Attributes	Invariants
Atom-Molecule	2	2	3	2
University	4	3	8	5
ATM Machine	50	51	51	7
Script 1	100	110	122	2
Script 2	500	510	522	5
Script 3	1000	1010	1022	5

Table 4. Description of the Examples.

Before Slicing				After Slicing			
Scope	TT	ST	TT+ST	STT	SST	STT+SST	Speedup %
2	9548ms	12941ms	22489ms	6ms	8ms	14ms	99.93%
3	9734ms	30041ms	39775ms	13ms	10ms	23ms	99.94%
4	12496ms	66861ms	79357ms	19ms	10ms	29ms	99.96%
5	15702ms	85001ms	100703ms	22ms	13ms	35ms	99.96%
6	19496ms	185118ms	204614ms	29ms	16ms	45ms	99.97%
7	23089ms	259072ms	282161ms	35ms	17ms	52ms	99.98%

TT Translation Time **ST** Solving Time
STT Sliced Translation Time **SST** Sliced solving Time

Table 10. Slicing results in Alloy for script 3 (1000 classes).


Introduction
Problem Statement
Research Objective
Proposed UOST
Results
Future Work

Currently Working On...

- Feedback technique in case of any unsatisfiable submodels.
- There are 3 stages of feedback technique:
  1. Detect the failed submodel.
  2. Detect the specific failed invariant.
  3. Provide constructive feedback for improvement.