

# Towards Compositional Approach to Model Transformation for Software Development

Soichiro Hidaka<sup>1</sup>, Zhenjiang Hu<sup>1</sup>, Hiroyuki Kato<sup>1</sup>,  
Keisuke Nakano<sup>2</sup>

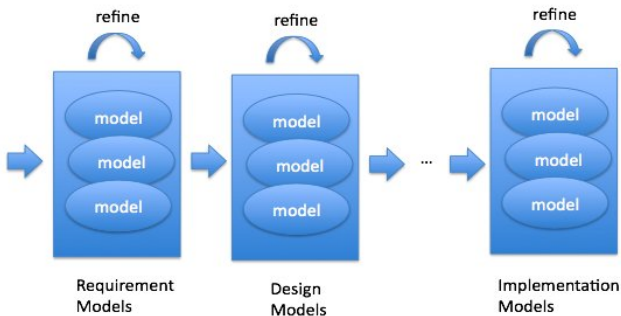
<sup>1</sup> National Institute of Informatics

<sup>2</sup> The University of Electro-Communications

September 12, 2008

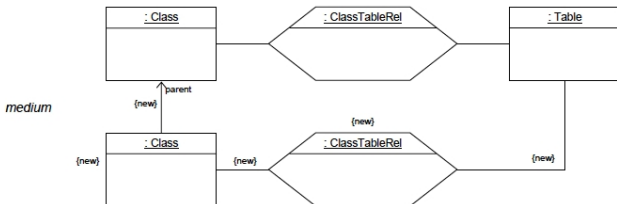
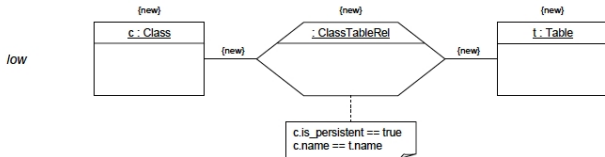
# Model-Driven Software Development

- Software Artifacts: Models
- Software Development: Model Transformations



- Graph-based Approach:
  - Models: Graphs
  - Model Transformations: Graph Transformations
  
- Existing Frameworks:
  - AGG: attributed graph grammars
  - TGG: triple graph grammars
  - QVT/ATL: Query/View/Transformation

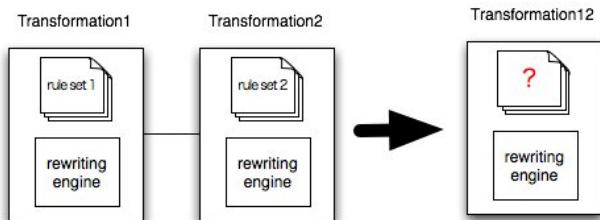
Triple Graphs: source graph + link graph + target graph



# Model Transformation Composition

The survey paper [Ehrig et al. 2005] says:

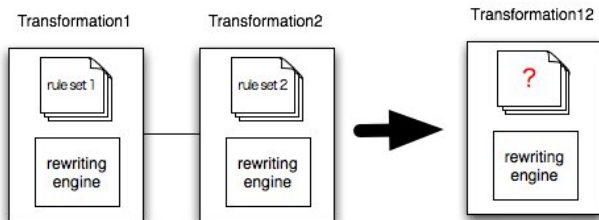
*Open issues* for all graph transformation approaches are elaborated concepts to compose transformations ...



# Model Transformation Composition

The survey paper [Ehrig et al. 2005] says:

*Open issues* for all graph transformation approaches are elaborated concepts to compose transformations ...



Composition is WANTED for systematic development of model transformation in the large [Klar et al: FSE07].

UnQL [Buneman et al. 2000] is a

compositional framework for graph querying

- Easy to use: `select ... where ...`
- Has a core `graph algebras` for arbitrary graph construction
- Use `structural recursion` for manipulating graphs

UnQL [Buneman et al. 2000] is a

compositional framework for graph querying

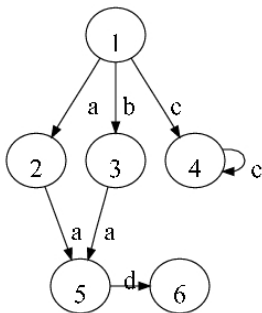
- Easy to use: `select ... where ...`
- Has a core `graph algebras` for arbitrary graph construction
- Use `structural recursion` for manipulating graphs

Can we extend UnQL  
from graph querying to graph transformation?

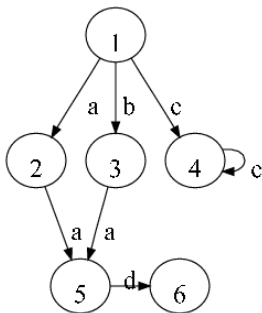
A compositional framework  $\text{UnQL}^+$  for model transformations:

- It is **functional**: not traditional "rule-based"  
⇒ the first general functional framework
- It is **algebraic**: graph algebras and structural recursion  
⇒ giving a clear semantics for model equivalence and model transformation.
- It is **practical**: systematic development of model transformation in large  
⇒ a new system has been implemented in OCaml (about 7,500 loc) and test with some nontrivial examples.

# Edge-labelled Graphs



# Edge-labelled Graphs



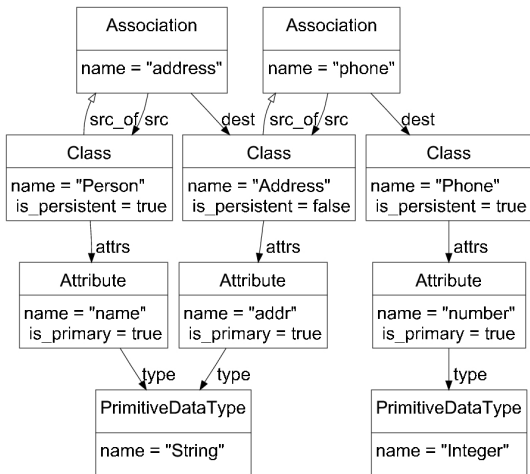
$$g = \{a : \{a : g_1\}, b : \{a : g_1\}, c : g_2\}$$

$$g_1 = \{d : \{\}\}$$

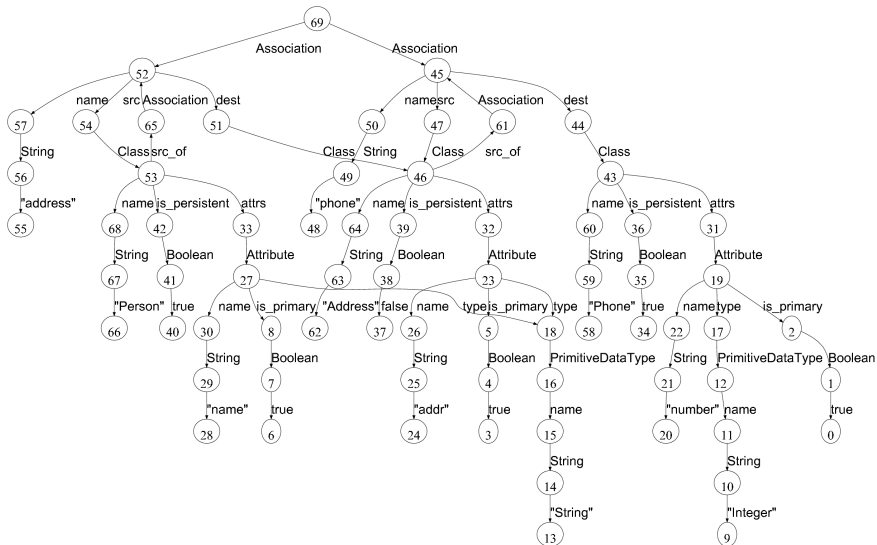
$$g_2 = \{c : g_2\}$$

# Model Representation: An Example

## A Class Diagram:



# Model Representation: An Example



# Structural Recursion: Manipulating Graphs

Structural Recursion:

$$\begin{aligned}f(\{\}) &= \{\}\\f(\{l : g\}) &= l \odot f(g) \\f(g_1 \cup g_2) &= f(g_1) \cup f(g_2)\end{aligned}$$

# Structural Recursion: Manipulating Graphs

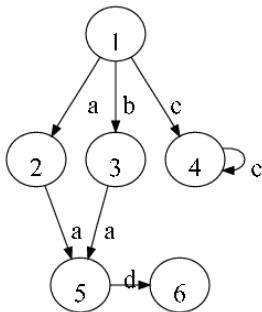
Structural Recursion:

$$\begin{aligned}f(\{\}) &= \{\}\\f(\{l : g\}) &= l \odot f(g)\\f(g_1 \cup g_2) &= f(g_1) \cup f(g_2)\end{aligned}$$

Or written as:

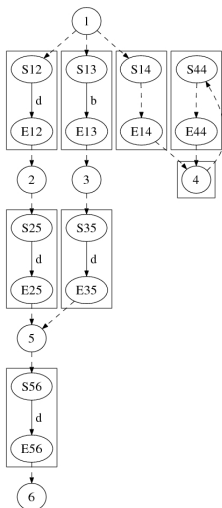
$$\text{sfun } f(\{l : g\}) = l \odot f(g)$$

# An Example

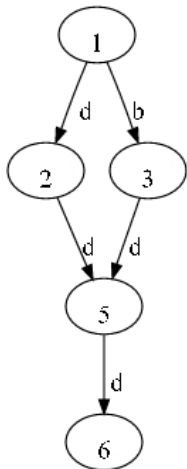
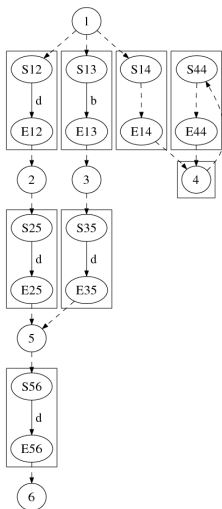


$\text{sfun } a2d\_xc (\{l : g\}) = \text{if } l = a \text{ then } \{d : a2d\_xc(g)\}$   
 $\text{else if } l = c \text{ then } a2d\_xc(g)$   
 $\text{else } \{l : a2d\_xc(g)\}$

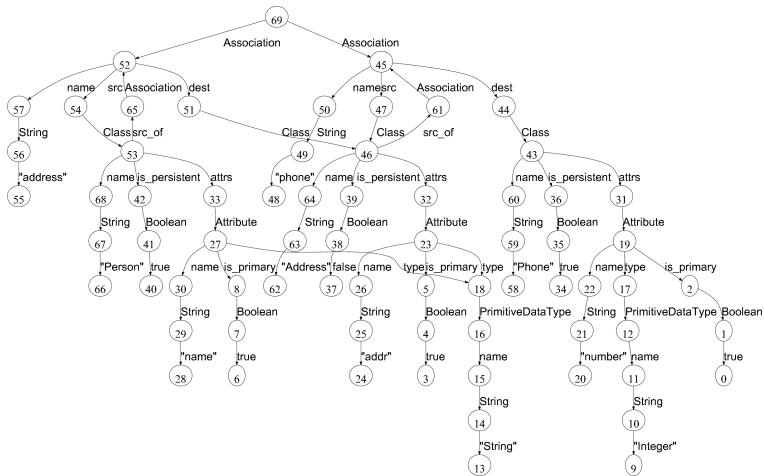
# The Bulk Semantics of Structural Recursion



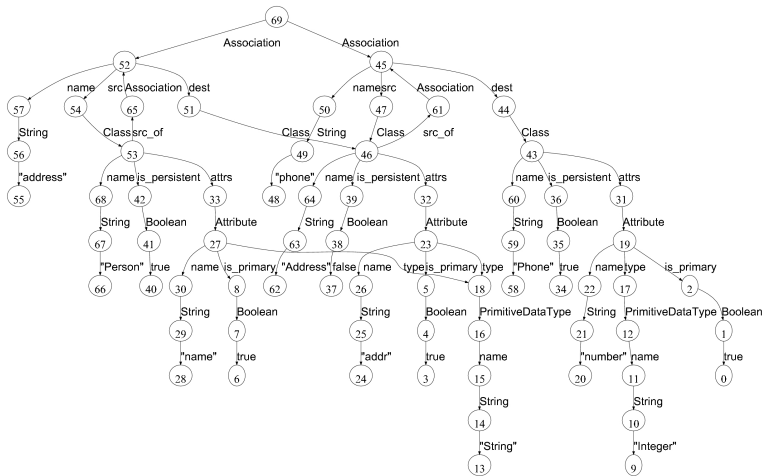
# The Bulk Semantics of Structural Recursion



# Graph Querying in UnQL



# Graph Querying in UnQL



How to extract all persistent classes?

**select** *\$class* **where**

```
{ Association.(src|dest).Class : $class } in $classDB,  
{ is_persistent : { Boolean : true } } in $class
```

```
select $class where  
  { Association.(src|dest).Class : $class } in $classDB,  
  { is_persistent : { Boolean : true } } in $class
```

Remarks:

- We do not need to use structural recursion explicitly!
- Any UnQL expression can be described as a composition of structural recursions.

# UnQL<sup>+</sup>: An Extension of UnQL

```
replace {$name : {}}  
by      {" class_" + $name} : {}}  
where  
  {_* .Class.name.String : {$name : {}}} in $classDB
```

# UnQL<sup>+</sup>: An Extension of UnQL

```
replace {$name : {}}  
by      {"class_" + $name} : {}}  
where  
  {_* .Class.name.String : {$name : {}}} in $classDB
```

## Property 1

Any expression in UnQL<sup>+</sup> can be described as a composition of structural recursions.

# UnQL<sup>+</sup>: An Extension of UnQL

```
replace {$name : {}}  
by      {" class_" + $name} : {}}  
where  
  {_* .Class.name.String : {$name : {}}} in $classDB
```

## Property 1

Any expression in UnQL<sup>+</sup> can be described as a composition of structural recursions.

## Property 2

Unnecessary compositions can be automatically removed.

# Transformation Compositions

Extract all persistent classes, and transform them to tables by replacing *attrs* by *cols* and *Attribute* by *Column*.

(\* replace *Attribute* \*)

replace{ $\$/A : \$/g$ } by{*Column* :  $\$/g$ } where  
 $\$/db$  in

(\* replace *attrs* \*)

(replace{ $\$/a : \$/A$ } by{*cols* :  $\$/A$ } where  
 $\$/class$  in

(\* select classes \*)

(select  $\$/class$  where

{*Association*.(*src|dest*).*Class* :  $\$/class$ }  
in  $\$/classDB$ ,

{*is\_persistent* : {*Boolean* : *true*}} in  $\$/class$ ),

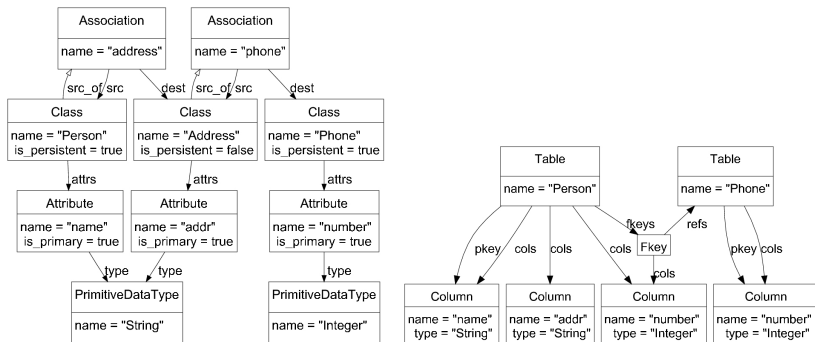
{ $\$/a : \$/A$ } in  $\$/class$ ,  $\$/a = attrs$ ),

{*cols* : { $\$/A : \$/g$ }} in  $\$/db$ ,  $\$/A = Attribute$



# Application: Class2RDB

Transformation from "Class to RDB", a nontrivial example proposed at MTiP'05.



```
select $tables where
  $tables in
    (select $tables where
      {Class:$class} in (select $assoc where {Association.(src|dest):$assoc} in $db),
      {is_persistent.Boolean:true} in $class,
      $dests in (select {Class:$dest} where {(src_of.Association.dest.Class)+:$dest} in $class),
      $related in ({Class:$class} U $dests),
      $cols in (select {cols:{Column:{name:$n,type:$t}}} where {Class.attrs.Attribute:{name:$n,type:$t}} in $related
      $tables in (select {Table:{name:$cname} U $cols} where {name:$cname} in $class),
      $tables in (extend $table with $pkeys U $fkeys where
        {Table:$table} in $tables,
        {cols:$cols} in $table,
        {Column.name.String:{$cname:{}}} in $cols,
        $pkeys in (select {pkey:$cols} where
          {attrs.Attribute: {is_primary.Boolean:true, name.String:{$pname:{}}}} in $class,
          $cname = $pname),
        $fkeys in (select {fkeys:{Fkey:{cols:$cols, ref:$ref}}} where
          {Class:{is_persistent.Boolean:true,
            attrs.Attribute.name.String:{$aname:{}}, name:$ref}} in $dests,
            $cname = $aname))),
      $tables in (replace $ref by {Table:$table} where
        {Table.fkeys.Fkey.ref:$ref, Table:$table} in $tables,
        {String:{$rname:{}}} in $ref,
        {name.String:{$tname:{}}} in $table,
        $tname = $rname)
```



We proposed the **first general functional framework** supporting systematic development of model transformation in a compositional manner.

- Easy to use (select-where, replace-where)
- Powerful to describe various model transformations
- Efficient to implement (fusion optimization)

<http://research.nii.ac.jp/~hidaka/big/www/>

We proposed the **first general functional framework** supporting systematic development of model transformation in a compositional manner.

- Easy to use (select-where, replace-where)
- Powerful to describe various model transformations
- Efficient to implement (fusion optimization)

<http://research.nii.ac.jp/~hidaka/big/www/>



compositional approach to **bidirectional model transformation**.