Towards Compostional Approach to Model Transformation for Software Development

Soichiro Hidaka\textsuperscript{1}, Zhenjiang Hu\textsuperscript{1}, Hiroyuki Kato\textsuperscript{1}, Keisuke Nakano\textsuperscript{2}

\textsuperscript{1} National Institute of Informatics  
\textsuperscript{2} The University of Electro-Communications

September 12, 2008
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
TGG/QVT: An Example

Triple Graphs: source graph + link graph + target graph

low

\[
\begin{align*}
&c: \text{Class} \\
&: \text{ClassTableRel} \\
&t: \text{Table}
\end{align*}
\]

\[
c screwed \text{Persistent} == \text{true} \]

\[
c \text{name} == t \text{name}
\]

medium

\[
\begin{align*}
&: \text{Class} \\
&: \text{ClassTableRel} \\
&: \text{Table}
\end{align*}
\]

\[
\begin{align*}
&\text{parent} \\
&: \text{Class}
\end{align*}
\]

\[
\begin{align*}
&: \text{ClassTableRel} \\
&\text{new}
\end{align*}
\]

\[
\begin{align*}
&: \text{Table}
\end{align*}
\]

Towards Compositional Approach to Model Transformation for So
The survey paper [Ehrig et al. 2005] says:

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

A compositional framework UnQL$^+$ for model transformations:

- It is **functional**: not traditional "rule-based"
  \[ \Rightarrow \text{the first general functional framework} \]

- It is **algebraic**: graph algebras and structural recursion
  \[ \Rightarrow \text{giving a clear semantics for model equivalence and model transformation.} \]

- It is **practical**: systematic development of model transformation in large
  \[ \Rightarrow \text{a new system has been implemented in OCaml (about 7,500 loc) and test with some nontrivial examples.} \]
Edge-labelled Graphs

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

\[ g_1 = \{ d: {} \} \]

\[ g_2 = \{ c: g_2 \} \]

Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, Keisuke Nakano

Towards Compositional Approach to Model Transformation for So
Edge-labelled Graphs

\[
g = \{ a : \{ a : g_1 \}, b : \{ a : g_1 \}, c : g_2 \} \\
g_1 = \{ d : \{ \} \} \\
g_2 = \{ c : g_2 \}
\]
A Class Diagram:

```
Association
  name = "address"
  src_of src

Class
  name = "Person"
  is_persistent = true
  attrs
    Attribute
      name = "name"
      is_primary = true
      type
        PrimitiveDataType
        name = "String"

Association
  name = "phone"
  src_of src

Class
  name = "Address"
  is_persistent = false
  attrs
    Attribute
      name = "addr"
      is_primary = true
      type
        PrimitiveDataType
        name = "String"

Class
  name = "Phone"
  is_persistent = true
  attrs
    Attribute
      name = "number"
      is_primary = true
      type
        PrimitiveDataType
        name = "Integer"
```
Model Representation: An Example

Towards Compositional Approach to Model Transformation for Software Development
Structural Recursion:

\[
\begin{align*}
  f (\{\}) & = \{} \\
  f (\{l : g\}) & = l \circ f(g) \\
  f (g_1 \cup g_2) & = f(g_1) \cup f(g_2)
\end{align*}
\]
Structural Recursion:

\[
\begin{align*}
  f(\{\}) & = \{} \\
  f(\{l : g\}) & = l \odot f(g) \\
  f(g_1 \cup g_2) & = f(g_1) \cup f(g_2)
\end{align*}
\]

Or written as:

\[
\text{sfun } f(\{l : g\}) = l \odot f(g)
\]
An Example

\[
\text{sfun } \text{a2d}_x c \left( \{l : g\} \right) = \begin{cases} 
\text{if } l = a \text{ then } \{d : \text{a2d}_x c(g)\} \\
\text{else if } l = c \text{ then } \text{a2d}_x c(g) \\
\text{else } \{l : \text{a2d}_x c(g)\}
\end{cases}
\]
The Bulk Semantics of Structural Recursion

Towards Compositional Approach to Model Transformation for Software Development
The Bulk Semantics of Structural Recursion

Towards Compositional Approach to Model Transformation for Software Development
Graph Querying in UnQL

How to extract all persistent classes?

Soichiro Hidaka¹, Zhenjiang Hu¹, Hiroyuki Kato¹, Keisuke Nakano²

Towards Compositional Approach to Model Transformation for Software Development
Graph Querying in UnQL

How to extract all persistent classes?

Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, Keisuke Nakano
select $class where

\{Association.(src|dest).Class : $class\} in $classDB,
\{is_presistent : \{Boolean : true\}\} in $class
Graph Querying in UnQL

select $class where
  {Association.(src|dest).Class : $class} in $classDB,
  {is_presistent : {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.
replace \{name : {}\} by \{("class" + name) : {}\} where \{_ ∗ .Class.name.String : {name : {}}\} in $classDB
**UnQL⁺: An Extension of UnQL**

replace `{\$name : {}}` by `{("class" + \$name) : {}}`

where

{\_\*.Class.name.String : \{$name : {}\}} in \$classDB

**Property 1**

Any expression in UnQL⁺ can be described as a composition of structural recursions.
replace \{ \$name : \{ \} \} 
by \{ ("class_" + \$name) : \{ \} \} 
where
\{ _\ast .Class.name.String : \{ \$name : \{ \} \} \} in \$classDB

**Property 1**

Any expression in UnQL\(^+\) can be described as a composition of structural recursions.

**Property 2**

Unnecessary compositions can be automatically removed.
Extract all persistent classes, and transform them to tables by replacing `attrs` by `cols` and `Attribute` by `Column`.

\[
(* \text{replace } \text{Attribute} \*)
\begin{align*}
\text{replace}\{l_A : g\} \text{ by } \{Column : g\} \text{ where} \\
\text{db} \text{ in} \\
(* \text{replace } \text{attrs} \*) \\
\text{(replace}\{l_a : A\} \text{ by } \{cols : A\} \text{ where} \\
\text{class} \text{ in} \\
(* \text{select classes} \*) \\
\text{(select } \text{class} \text{ where} \\
\{\text{Association.}(src|\text{dest}).\text{Class} : \text{class}\} \\
\text{ in } \text{classDB}, \\
\{\text{is_persistent : } \{\text{Boolean} : \text{true}\}\} \text{ in } \text{class}), \\
\{l_a : A\} \text{ in } \text{class}, l_a = \text{attrs}, \\
\{cols : \{l_A : g\}\} \text{ in } \text{db}, l_A = \text{Attribute}
\end{align*}
\]
(7,500 lines of code in OCaml)
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 $keys U $keys where
      {Table:$table} in $tables,
      {cols:$cols} in $table,
      {Column.name.String:$cname} in $cols,
      $keys in {select {pkey:$cols} where
        {attrs.Attribute: {is_persistent:Boolean:true, name.String:$name} in $class,
         $name = $pkey},
      $keys in {select {fkeys:{Fkey:{cols:$cols, ref:$ref}}} where
        {Class:{is_persistent:Boolean:true,
          attrs.Attribute.name.String:$name} in $dests,
         $name = $fkey},
      $tables in {replace $ref by {Table:$table} where
        {Table.fkeys.Fkey.ref:$ref, Table:$table} in $tables,
        {String:$name} in $ref,
        {name.String:$tname} in $table,
        $tname = $name
    )}}
)}

Soichiro Hidaka¹, Zhenjiang Hu¹, Hiroyuki Kato¹, Keisuke Nakai²
Towards Compostional Approach to Model Transformation for Software Development
Towards Compositional Approach to Model Transformation for Software Development
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.