Source Transformation Systems Supporting Software Engineering

Magne Haveraaen
University of Bergen
University of Wales Swansea
together with Bagge, Kalleberg a.o.

STS'04
part of GPCE'04

Vancouver, Canada, October 2004

This research supported in part by the Research Council of Norway

Notational Variations

Too many syntactic variations of one method idea in most programming languages

- Functional form

```
a := sort(b);          e := getElement(b,i);      b := setElement(b,i,e);
a := b.sort();         e := b[i];                 b[i] := e;
```

- Imperative form

```
sort(b,a); sort(a,b);
a.sort(b); b.sort(a);
```

- Imperative, in situ form

```
sort(a);
a.sort();                                         b.setElement(i,e);
```

- Program form

```
unix% sort b > a
```

Solution

- Normalise notation for user

```
a := sort(b);          e := b[i];                 b[i] := e;
```

- Allow full freedom for implementor

```
a.sort();              b.getElement(i,e);         b.setElement(i,e);
```

STS provides link

- Envelope implemented code in standardised functional/program form
- Rewrite from user invoked form to actually implemented form (*mutification*)

User defined / Domain Specific Optimisation Rules

Allow user to write simple code

```
// central value ignoring lower and upper quartile
r := ( sort(b)[length(b)/4.0] + sort(b)[3*length(b)/4.0] ) / 2.0;
```

Transform code using userdefined rewrite rules and mutification

```
r := ( sort(b)[length(b)/4.0] + sort(b)[3*length(b)/4.0] ) / 2.0;
```

↓

```
r := ( partition(b,length(b)/4.0) + partition(b,3*length(b)/4.0) ) / 2.0;
```

↓

```
a1 := partition(b,length(b)/4.0);
a2 := partition(b,3*length(b)/4.0);
r := ( a1+a2 ) / 2.0;
```

↓

```
Partition b'.setUpPartition(b);
b'.partition(length(b)/4.0,a1); b'.partition(3*length(b)/4.0,a2);
r := ( a1+a2 ) / 2.0;
```

User defined rewrite rules

- sort(b)[x] → partition(b,x)
- a1 := partition(b,x); a2 := partition(b,y);
  → Partition b'.setUpPartition(b); b'.partition(x,a1); b'.partition(y,a2);

Summary

Source Transformation Systems may

- Easily support
  - Mutiple notations for method calls
  - Methods as programs
- Exploit software specifications as rewrite rules for
  - optimisations
  - views of semantically related methods
- Provide software organisation principles across programming languages
  - ADT and class encapsulation
  - invasive composition

All this gives substantial software engineering benefits – at low costs?

Can we develop STS as language independent SE support systems?