A Preliminary Study of Quantified, Typed Events

Robert Dyer[1], Mehdi Bagherzadeh[1], Hridesh Rajan[1] and Yuanfang Cai[2]

[1]Iowa State University
{rdyer,mbagherz,hridesh}@cs.iastate.edu

[2]Drexel University
yfcai@cs.drexel.edu

March 16, 2010

Ptolemy: What, Why?

- Ptolemy[1] adds quantified, typed events to OO languages
  1. Well-defined interfaces between base & crosscutting code
  2. Separate type-checking, modular reasoning
- Combines aspect-oriented (AO) and implicit invocation (II)
- Solves problems with AO and II:
  - AO: quantification failure, fragile pointcuts, limited context information
  - II: coupling of observers, no replacement of event code, no quantification

[1] Rajan and Leavens - ECOOP'08

This Paper: Why, How, and What?

- Motivation: Why use Quantified, Typed Events[2]?
- Approach: MobileMedia case study[3]
- Evaluation: Change impact and Design value analysis
  - Software engineering metrics: makes implicit coupling in AO explicit and decreases change impact
  - NOV analysis: Ptolemy needs ITDs (so we added it)

[2] Rajan and Leavens - ECOOP'08
[3] Figueiredo et al - ICSE'08

Running Example : Figure Editor

- Elements of drawing
  - Points, Lines, etc
  - All such elements are of type `FElement`
- **Challenge:** Modularize display update policy
  - Whenever an element of drawing changes —
  - Update the display

Point and its two Events

```
class Point implements FElement {
  int x; int y;
  void setX(int x) {
    this.x = x;
  }
  ..
  void makeEqual(Point other) {
   if(!other.equals(this)) {
     other.x = this.x;
     other.y = this.y;
}}}
```

- Changing FElement is different for two cases.
- Actual abstract event inside `makeEqual` is the true branch.

Aspect Modularizing Display Updating

```
aspect Update {
  around(FElement fe) :
    execution(Point.set*(..)) && this(fe) ||
    (execution(Point.make*(..)) && args(fe)
     if(!fe.equals(this(fe)))) {
    proceed(fe);
    Display.update();
}}
```

- Enumeration required of two different joinpoints.
- Had to use **`if`** pointcut to get to the real event.
- Alternative is to refactor `makeEqual` (refactoring doesn't always creates meaningful abstractions).

Ptolemy: Declaring Event Types

```
void event FEChanged {
  FElement changedFE;
}
```

- Event type is an abstraction (design this first).
- Declares context available at the concrete events.
- Interface, so allows design by contract (DBC) methodology.

Ptolemy: Announcing Events

```
class Point implements FElement {
  int x; int y;
  void setX(int x) {
    announce FEChanged(this) {
      this.x = x;
}}
  void makeEqual(Point other) {
   if(!other.equals(this)) {
    announce FEChanged(other) {
      other.x = this.x; other.y = this.y;
}}}}
```

- Explicit, declarative, typed event announcement.
- Provides flexibility, e.g. see `makeEqual`.

Ptolemy: Binding to Events

```
class Update {
  when FEChanged do update;
  void update (FEChanged next) {
    invoke(next); //Like AspectJ proceed
    Display.update();
  }
  public Update() {
    register(this); //Allows dynamic deployment
}}
```

Research Questions

- How do these two designs compare?
- When do we see benefits of AO?
- When do we see benefits of Ptolemy?

Observed Benefits of Aspect-oriented Designs

- Static crosscutting features are very useful
  - Inter-type declarations (ITDs)
  - Declare Parents
  - Softened Exceptions

Inter-type declarations (ITDs)

- Had to be emulated in Ptolemy[4]
- AspectJ:

```
public T C.field;
```

- Ptolemy emulation strategy:

```
static Hashtable fieldMap;
public static T getField(C);
public static void setField(C, T);
```

[4] Ptolemy now supports AspectJ-style ITDs

Declare Parents

- Affects the type hierarchy
- Only used in revision 8
- Effects modeled similar to ITDs

Softened Exceptions

- Aspects handle certain exceptions
- Softened exceptions don't need declared thrown in the base code

- Con: Ptolemy version still must declare those exceptions are thrown
- Pro: Ptolemy's exception handling code is (un)pluggable

Softened Exceptions

AspectJ version:

```
declare soft: RecordStoreEx :
  execution(public void
    ImageAccessor.addImageData(..));
```

```
public void addImageData(..) throws
  InvalidImageDataEx, PersistenceMechanismEx {
```

Without the aspect, the base code won't compile!

Softened Exceptions

Ptolemy version:

```
public void addImageData(..) throws
  InvalidImageDataEx, PersistenceMechanismEx,
  RecordStoreEx {
```

Even though RecordStoreEx isn't thrown by the body, it still must be declared!

- Observed Benefits of Quantified, Typed Events
  - No Quantification Failure
  - No Fragile Pointcuts
  - Can easily advise other advice (due to symmetry)

Solves Quantification Failure Problem

- In revision 2, the AspectJ version had to expose a **`while`** loop (by refactoring of course)
- Similar problems in other revisions
- Ptolemy versions did not need to refactor those points to expose to the aspects

Example of Quantification Failure in AO

OO version:

```
..
while (is.read(b)) { .. }
..
```

AspectJ version:

```
..
internalReadImage(..);
..

private void internalReadImage(..) {
  while (is.read(b)) { .. }
}
```

No Quantification Failure

OO version:

```
..
while (is.read(b)) { .. }
..
```

Ptolemy version:

```
..
announce ReadInternalImageAsByteArrayEvent() {
  while (is.read(b)) { .. }
}
..
```

No Fragile Pointcuts

- Aspects implicitly match the base code
- Quantified, typed events make this coupling explicit
- Changes to the base code (e.g., renaming a method) can propogate to the aspects

- AspectJ: `execution(* DeletePhoto(..))`
- Ptolemy: `when DeletePhotoEvent do Handler`
- What if `DeletePhoto` is renamed to `RemovePhoto`?

Can easily advise other advice

- AspectJ allows you to advise all advice, not specific advice bodies
- Exception handling modularity was not maintained in later revisions for AspectJ versions
- Ex: revision 8, aspect lancs.midp.mobilephoto.alternative.music.MusicAspect

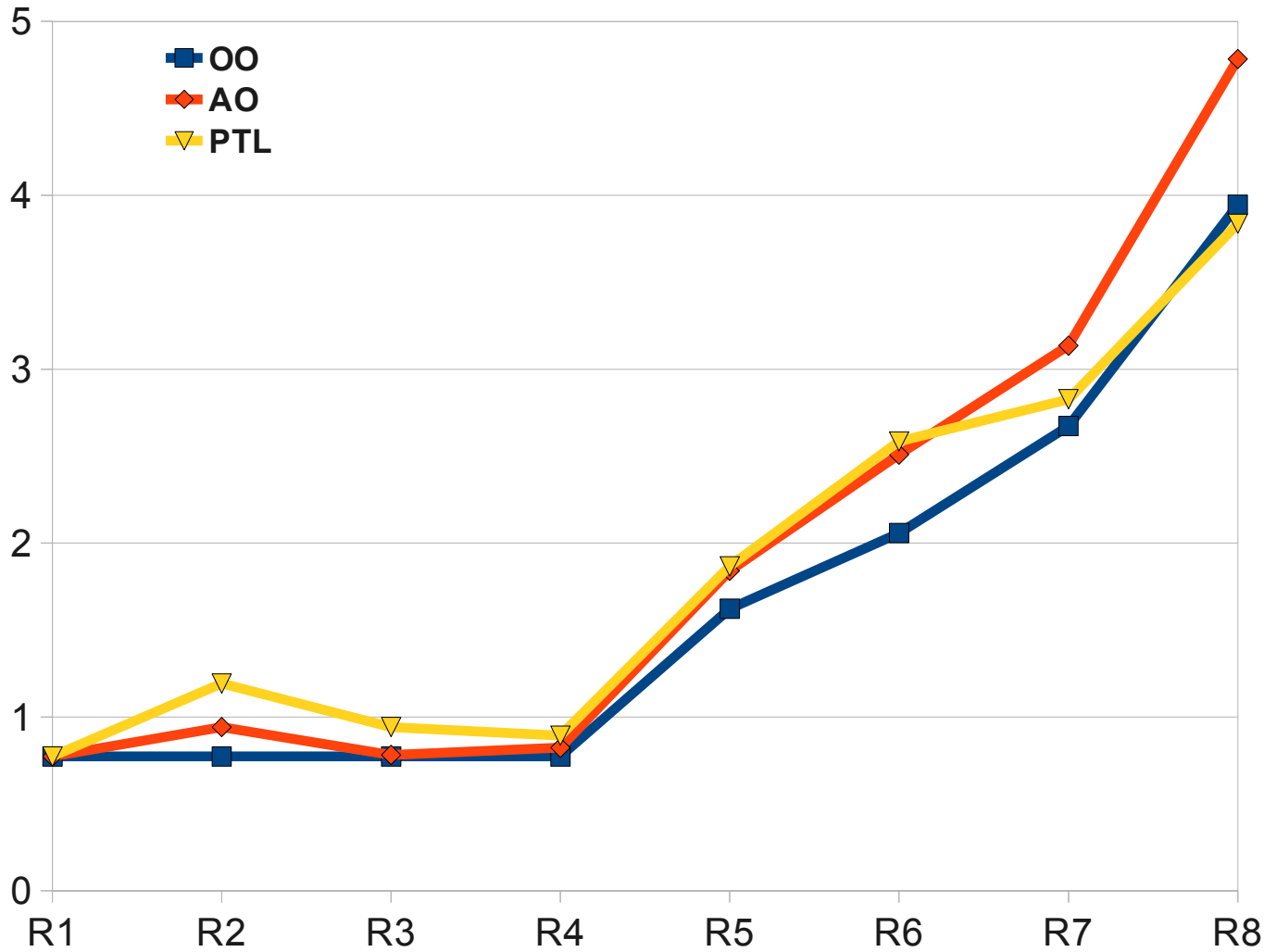
```
after() : addNewMediaToAlbum() {
  try {
    /* advice body */
  } catch (InvalidImageDataException e) {
    ..
  }
}
```

```
public void handler(AddMediaToAlbumEvent next){
  announce AddNewMediaToAlbumHandlerEvent() {
    /* advice body */
  }
}
```

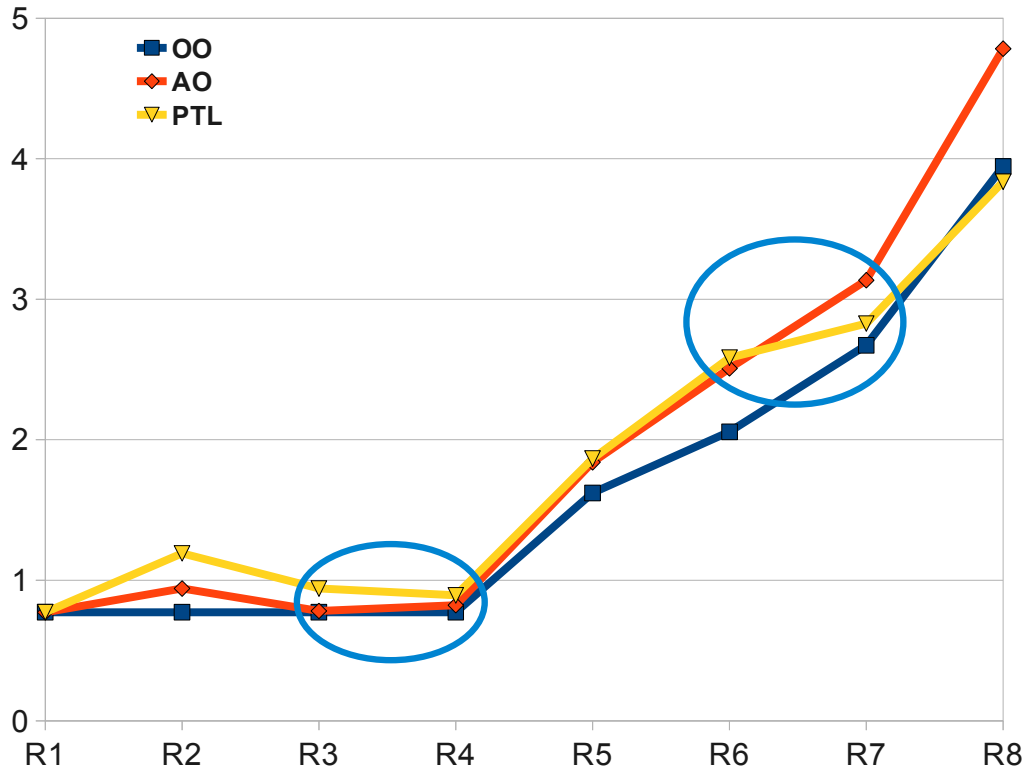
Overview of Change Impact AO vs. Ptolemy

- Ptolemy design limited change propagation
- Only had to change 13 event types in revision 7
- AO revisions required changing 50 pointcuts in revision 7
- No other Ptolemy revision required changing event types
- 28 pointcuts changed across 4 other AO revisions

Net Options Value Analysis AO vs. Ptolemy



Net Options Value Analysis AO vs. Ptolemy



- Ptolemy: higher NOV values for R2-R6
- AO: higher value in R6 (Ptolemy doesn't have ITDs)
- Revisions adding ITDs: R3-R8

Summary of Study and Future Work

- Explicit Coupling increases from OO $\longrightarrow$ AO $\longrightarrow$ Ptolemy
- Despite more coupling in Ptolemy, lower change impact

- Net options value increases from OO $\longrightarrow$ AO $\longrightarrow$ Ptolemy

Future Work:

- Repeat the study with Ptolemy + ITDs
- Compare to other AO interface features

Questions?

http://www.cs.iastate.edu/~ptolemy/
http://ptolemyj.sourceforge.net

Release	Description	Type of Change
R1	MobilePhoto core	
R2	Exception handling included	Non-functional concern
R3	Added photo sorting Added editing photo labels	Optional feature Mandatory feature
R4	Added favorites	Optional feature
R5	Allow users to keep multiple copies of photos	Optional feature
R6	Added send photo to other users by SMS	Optional feature
R7	Photo management made into two alternatives: photo or music	One mandatory feature into two alternatives
R8	Add video management	Alternative feature

			R2	R3	R4	R5	R6	R7	R8
Components	Added	OO	9	1	0	5	7	17	6
		AO	12	2	3	6	8	21	16
		PTL	13	4	2	6	8	23	18
	Removed	OO	0	0	0	0	0	10	1
		AO	1	0	0	0	0	8	0
		PTL	1	1	0	1	0	7	2
	Changed	OO	5	8	5	8	6	12	22
		AO	5	10	2	8	5	16	9
		PTL	11	8	1	9	5	16	8
PCs	Added	AO	43	6	7	2	7	19	26
	Removed	AO	0	0	0	0	0	0	5
	Changed	AO	0	8	0	16	2	50	2
Event Types	Added	PTL	25	9	1	5	5	9	4
	Removed	PTL	0	1	0	1	0	3	0
	Changed	PTL	0	0	0	0	0	13	0

			R2	R3	R4	R5	R6	R7	R8
LCOO	Average	OO	6.38	10.08	10.88	8.83	10.24	12.04	11.80
		AO	5.67	8.69	8.50	6.97	8.24	9.39	8.03
		PTL	3.30	3.36	3.21	2.86	2.83	2.77	2.96
	Max	OO	64	70	71	73	96	113	114
		AO	64	94	94	64	85	109	111
		PTL	84	122	122	66	66	112	153
CBC	Average	OO	1.46	2.00	2.36	3.17	3.30	3.54	3.96
		AO	1.30	1.72	1.84	2.50	2.65	2.76	2.69
		PTL	0.80	0.94	1.06	1.43	1.57	1.71	2.01
	Max	OO	9	13	13	11	11	15	20
		AO	9	13	12	10	13	14	14
		PTL	9	14	14	12	13	14	14

		R2	R3	R4	R5	R6	R7	R8
LOC	OO	1159	1314	1363	1555	2051	2523	3016
	AO	1276	1494	1613	1834	2364	3068	3806
	PTL	1605	1923	2049	2374	2969	3655	4508
NOC	OO	24	25	25	30	37	46	51
	AO	27	29	32	38	46	59	75
	PTL	56	67	70	79	92	112	132
NOA	OO	62	71	74	75	106	132	165
	AO	62	72	76	77	111	139	177
	PTL	72	82	86	88	121	146	185
NOO	OO	124	140	143	160	200	239	271
	AO	158	187	199	230	285	345	441
	PTL	143	169	179	197	247	308	369

1 Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sergio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho and Francisco Dantas. Evolving software product lines with aspects: an empirical study on design stability. In ICSE '08.

2 Hridesh Rajan and Gary T. Leavens. Ptolemy: a language with quantified, typed events. In ECOOP '08.

4 Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, and Ben Hallen. The structure and value of modularity in software design. In ESEC/FSE '01.