Ptolemy

Balances expressiveness and modular reasoning for aspect-oriented software development.

Hridesh Rajan, Gary T. Leavens, Sean Mooney, Rex D. Fernando, Robert Dyer, and Mehdi Bagherzadeh

ptolemy.cs.iastate.edu
Tutorial Outline

❖ Why Ptolemy? What problems does it solve?
  ❖ Two precursors
    ➢ Implicit Invocation and Aspect-orientation

❖ Ptolemy and how it solves these problems.

❖ Main Language Features
  ❖ Declarative, typed events (join points in AO terms)
  ❖ Declarative, typed event announcement (no AO term)
  ❖ Declarative, typed event registration (advising in AO terms)
  ❖ Quantification based on event types (same as the AO term)
Tutorial Outline

❖ Modular Verification Features
  ❖ Translucid Contracts (no AO term)

❖ Where to use Ptolemy Features?
  ❖ vs. Aspect-orientation,
  ❖ vs. Implicit Invocation

❖ State of Tools

❖ Opportunities to Contribute

❖ Conclusion
One shall not have to choose between reasoning and separation.

WHY PTOLEMY?
Color-coded representation of about 19K LOC at ASML: different colors represent different concerns in the system.

Courtesy: Bruntink, Deursen and Tourné
Need for Improved Separation

- Some concerns hard to modularize
- Number of proposals: Units [Flatt and Felleisen], Mixin [Bracha and Cook], Open Classes [Clifton et al.], Roles [Kristensen and Osterbye], Traits [Scharli et al.], Implicit Invocation [Garlan, Notkin, Sullivan et al.], Hyperslices [Ossher and Tarr], Aspects [Kiczales et al.], etc
- Shows that there is a real need
Two similar ideas

- Implicit invocation (II) vs. Aspect-orientation (AO)
- … both effective for separation of concerns
- … both criticized for making reasoning hard
  - II criticized in early/late 90’s
  - AO criticized in early 2000’s

- Ptolemy is designed to
  - combine best ideas from II and AO
  - … and to make reasoning easier
[JHotDraw – Gamma et al.]

RUNNING EXAMPLE
Elements of a Drawing Editor

- Elements of drawing
  - Points, Lines, etc
  - All such elements are of type Fig

- Challenge I: Modularize display update policy
  - Whenever an element of drawing changes — Update the display

- Challenge II: Impose application-wide restriction
  - No element may move up by more than 100
Figure Elements

1 abstract class Fig {
2 }

❖ Fig – super type for all figure elements
  ❖ e.g. points, lines, squares, triangles, circles, etc.
Point and its Two Events

1. class Point extends Fig {
   2    int x;
   3    int y;
   4    void setX(int x) {
   5       this.x = x;
   6    }
   7    ..
   8    void makeEqual(Point other) {
   9       if(!other.equals(this)) {
  10          other.x = this.x;
  11          other.y = this.y;
  12     }}
}

- Changing Fig is different for two cases.
- Actual abstract event inside makeEqual is the true branch.
Reiss’92, Garlan and Notkin’92

IMPLICIT INVOCATION
Key Ideas in II

❖ Allow management of name dependence
  ❖ when “Point’s coordinates changes” update Display
  ❖ ... but Point shouldn’t depend on Display
  ❖ ... complicates compilation, test, use, etc

❖ Components (subjects) declare events
  ❖ e.g. when “Point’s coordinates changes”
  ❖ provide mechanisms for registration
  ❖ ... and for announcement

❖ Components (observers) register with events
  ❖ e.g. invoke me when “Point’s coordinates changes”

❖ Subjects announce events
  ❖ e.g. when “Point’s coordinates changes”
  ❖ “change in coordinates” event announced
II: Components Declare Events

**abstract class** Fig {  
  List changeObservers;  
  **void** announceChangeEvent(Fig changedFE) {  
    **for** (ChangeObserver o : changeObservers) {  
      o.notify(changedFE);  
    }  
  }  
  **void** registerWithChangeEvent(ChangeObserver o) {  
    changeObservers.add(o);  
  }  
}  

**abstract class** ChangeObserver {  
  **void** notify(Fig changedFE);  
}
II: Components Announce Events

```java
1 class Point extends Fig {
2   int x; int y;
3   void setX(int x) {
4     this.x = x;
5     announceChangeEvent(this);
6   }
7   void makeEqual(Point other) {
8     other.x = this.x; other.y = this.y;
9     announceChangeEvent(other);
10  }
11 }
```

- Event announcement explicit, helps in understanding
- Event announcement flexible, can expose arbitrary points
**II: Component Register With Events**

```java
1 class Update extends ChangeObserver {
2   Fig last;
3   void registerWith(Fig fe) {
4     fe.registerWithChangeEvent(this);
5   }
6   void notify(Fig changedFE){
7     this.last = changedFE;
8     Display.update();
9   }
10 }
```

- Registration explicit and dynamic, gives flexibility
- Generally deregistration is also available
II: Disadvantages

- Coupling of observers to subjects
  
  ```java
  void registerWith(Fig fe) {
      fe.registerWithChangeEvent(this); ...
  }
  ```

- Lack of quantification
  
  ```java
  void registerWith(Point p) {
      p.registerWithChangeEvent(this);
  }

  void registerWith(Line l) {
      l.registerWithChangeEvent(this);
  }
  ```
II: Disadvantages

❖ No ability to replace event code

```java
class MoveUpCheck extends ... {
    void notify(Fig targetFE, int y, int delta) {
        if (delta < 100) { return targetFE }
        else{throw new IllegalArgumentException()}
    }
}
```
Kiczales et al. 97, Kiczales et al. 2001

ASPECT-BASED SOLUTIONS
Key Similarities/Differences with II

❖ Events ≡ “join points”
  ❖ AO: pre-defined by the language/ II: programmer
  ❖ AO: Implicit announcement/ II: explicit

❖ Registration ≡ Pointcut descriptions (PCDs)
  ❖ AO: declarative

❖ Handlers ≡ “advice” register with sets of events

❖ Quantification: using PCDs to register a handler with an entire set of events
Aspect-based Solution

1 aspect Update {
2     Fig around(Fig fe) :
3          call(Fig+\texttt{.set*}(..)) && target(fe)
4            || call(Fig+\texttt{.makeEq*}(..)) && args(fe){
5             Fig res = proceed(fe);
6             Display.update();
7             return res;
8 }
Advantages over II

❖ Ease of use due to quantification

❖ By not referring to the names, handler code remains syntactically independent
Limitations: Fragility & Quantification

- **Fragile Pointcuts:** consider method “settled”

  ```java
  Fig around(Fig fe) {
      call(Fig+.set*(..)) && target(fe)
      || call(Fig+.makeEq*(..)) && args(fe){
      ...
  }
  ```

- **Quantification Failure:** Arbitrary events not available

  ```java
  Fig setX(int x){
      if (x.eq(this.x)) { return this; }
      /* abstract event change */
      else { this.x = x; return this; }
  }
  ```
Limitations: Context access

⚠ Limited Access to Context Information

⚠ Limited reflective interface (e.g. “thisJoinPoint” in AJ)

⚠ Limited Access to Non-uniform Context Information

1 Fig around(Fig fe) :
2 call(Fig+.set*[..]) && target(fe)
3 || call(Fig+.makeEq*[..]) && args(fe){
4  ...

Limitations: Pervasive Join Point Shadows

- For each join point shadow, all applicable aspect should be considered (whole-program analysis)

```java
1 x = o1.m1(a.e1(), b.e2());
2 y = o2.m2(c.e3(), x);
```
Ptolemy (Claudius Ptolemaeus), fl. 2d cent. A.D., celebrated Greco-Egyptian mathematician, astronomer, and geographer.
Evolution of the Ptolemy Language

HyperJ
[Ossher, Tarr, Harrison 2001]

AspectJ
[Kiczales et al. 2001]

Eos
[Rajan and Sullivan 2003, 2005]

XPI
[Sullivan et al. 2005]

XPI - AspectJ
[Griswold et al. 2006]
Design Goals of Ptolemy

❖ Enable modularization of crosscutting concerns, while preserving encapsulation of object-oriented code,

❖ enable well-defined interfaces between object-oriented code and crosscutting code, and

❖ enable separate type-checking, separate compilation, and modular reasoning of both OO and crosscutting code.
First and foremost

- Main feature is event type declaration.

- Event type declaration design similar to API design.
  - What are the important abstract events in my application?
  - When should such events occur?
  - What info. must be available when such events occur?

- Once you have done it, write an event type declaration.
Declaring an Event Type

Fig `event` Changed {
  Fig fe;
}

Event Type Declaration
Declaring an Event Type

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

---

**Subject**

```java
1 class Fig {bool isFixed;}
2 class Point extends Fig{
3     int x, y;
4     Fig setX(int x){
5         announce Changed(this){
6             this.x = x; return this;
7         }
8     }
9 }
```

- Explicit, more declarative, typed event announcement.
More Event Announcements

• Explicit, more declarative, typed event announcement.

```java
class Point extends Fig{
    Fig moveUp(int delta){
        announce MoveUpEvent(this){
            this.y += delta; return this;
        }
    }
}
```
Advising Events

- No special type of “aspect” modules
- Unified model from Eos [Rajan and Sullivan 2005]

```java
class DisplayUpdate {

observer(Handler)
}
```
Quantification Using Binding Decls.

- Binding declarations
  - Separate “what” from “when” [Eos 2003]

```java
class DisplayUpdate {

    when Changed do update;

}
```

Observer (Handler)
Dynamic Registration

❖ Allow dynamic registration
❖ Other models can be programmed

```java
class DisplayUpdate {

    void DisplayUpdate() { register(this) }

    Fig update(Changed next) {

    }

    when Changed do update;

}
```

Observer(Handler)

Registration

Quantification
Controlling Overriding

- Use invoke to run the continuation of event
- Allows overriding similar to AspectJ

```java
class DisplayUpdate {
    void DisplayUpdate() { register(this); }

    Fig update(Changed next) {
        System.out.println("Before Invoke");
        next.invoke();
        Display.update();
        System.out.println("After Invoke");
    }

    when Changed do update;
}
```
Exercise 0: Get the distribution

❖ Go to the URL to download Ptolemy1.2 Beta5

http://ptolemy.cs.iastate.edu/ptolemy-ase11.zip

to download the zip file ptolemy-ase-11.zip

❖ Unzip the contents at a convenient location, while preserving its directory structure

❖ Start Eclipse and change your workspace to

$Download_Location$/pyc/workspace
Exercise 1: Figure Editor Example

- Browse code in 00-FigureExample Project
- [a] Open file FEChanged.java
  - Note return type and context variables of event declaration FEChanged
- [b] Open file Point.java
  - Note event announcements in setX, setY, moveBy
  - Note different context in method makeEqual.
    - Everywhere else context variable changedFE is bound to this, but in this method it is bound to other.
Exercise 1: Figure Editor Example

- [c] Open file DisplayUpdate.java
  - Note the binding declaration
    - when FEChanged do update

- Note the `register` statements
  - `register (..)`
  - It registers the receiver object to listen to events mentioned in the binding declarations
EXTENDED EXAMPLE
Expressions and Operations

ASTNodes

\[ e ::= v \]
\[ \quad | \text{(lambda } (v) \cdot e \text{ )} \]
\[ \quad | (e \ e) \]

\[ e ::= \ldots \]
\[ \quad | \text{true } | \text{false } | \text{Num} \]
\[ \quad | e == e | e <= e \]
\[ \quad | e && e | e `|` | e \]
\[ \quad | e + e | e * e | e - e \]

Eval
\[ E: e ==> e' \]

Checker
\[ T |-- e : t \]

Printer
Goal: Separation of Concerns

ASTNodes

\[ e ::= v \]
\[ \quad | (\text{lambda} (v) \ . \ e) \]
\[ \quad | (e \ e) \]

\[ e ::= \ldots \]
\[ \quad | \text{true} \ | \text{false} \ | \text{Num} \]
\[ \quad | e = e \ | e \leq e \]
\[ \quad | e \&\& e \ | e`|\|e \]
\[ \quad | e+e \ | e*e \ | e-e \]

Eval
\[ E : e \implies e' \]

Checker
\[ T \vdash e : t \]

Printer
Goal: Separation of Operations

### AST Nodes

\[ e ::= v \]
\[ \text{lambda } (v) . e \]
\[ (e \ e) \]

\[ e ::= ... \]
\[ \text{true} | \text{false} | \text{Num} \]
\[ e == e | e <= e \]
\[ e &\& e | e` | e` | e \]
\[ e + e | e * e | e - e \]

### AST Events

- Eval: \( E: e \Rightarrow e' \)
- Checker: \( T |-- e : t \)
- Printer
Enabling modular verification

CONTRACTS IN PTOLEMY
DEMO
Conclusion

- Motivation: intellectual control on complexity essential
  - Implicit invocation (II) and aspect-orientation (AO) help
  - ... but have limitations

- Ptolemy: combine best ideas of II and AO
  - Quantified, typed events + arbitrary expressions as explicit events
  - Translucid contracts

- Benefits over implicit invocation
  - decouples observers from subjects
  - ability to replace events powerful

- Benefits over aspect-based models
  - preserves encapsulation of code that signals events
  - uniform and regular access to event context
  - robust quantification

- Last but not least, more modular reasoning
Opportunities to Contribute

❖ Language design efforts
  ❖ Ptolemy# to come out in June, testing underway (Extension of C#)
  ❖ Transition to less front-end changes (for PtolemyJ)

❖ Verification efforts
  ❖ More expressive support for embedded contracts
  ❖ Practical reasoning approaches for heap effects
  ❖ Better verification error reporting
Opportunities to Contribute

❖ Case study efforts – compiler supports metrics
  ❖ Showcase applications, examples for Ptolemy
  ❖ Comparison with other languages/approaches

❖ Infrastructure efforts
  ❖ Support in Eclipse, other IDEs
  ❖ Better error reporting, recovery

❖ Language manuals, descriptions, …

All are welcome!!!

Open source MPL 1.1 License
I really need to separate my crosscutting concerns.

SURE, but can you give up modular reasoning?

NO WAY!

I can redefine reasoning for you.

NO WAY!

Language Expert 1

I desperately need to separate my crosscutting concerns.

I DON'T THINK SO! Didn’t you hear that AOP is BAD for reasoning?

Pretty much.

SO I AM STUCK!

Language Expert 2

I can redefine reasoning for you.

SO I AM STUCK!

Developer

NO WAY!

Day 1

Day 2

Balances expressiveness and modular reasoning for aspect-oriented software development.

Hridesh Rajan, Gary T. Leavens, Sean Mooney, Rex D. Fernando, Robert Dyer, and Mehdi Bagherzadeh

ptolemy.cs.iastate.edu