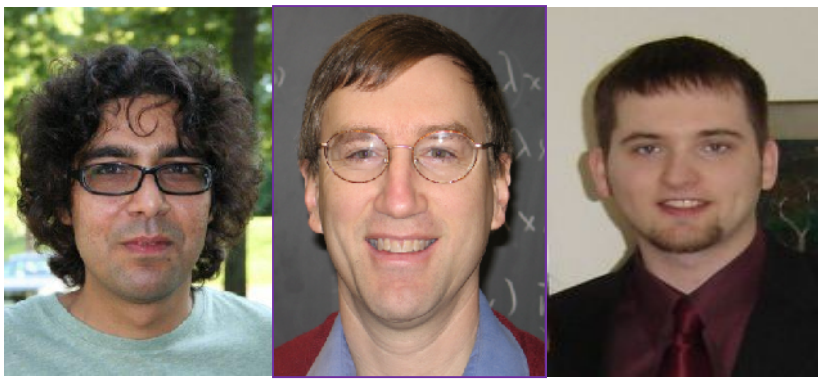
Applying Translucid Contracts for Modular Reasoning about Aspect and Object Oriented Events



Mehdi Bagherzadeh
Gary T. Leavens
Robert Dyer

Foundations of Apsect Oriented Languages 2011
Porto de Galinhas, Pernambuco, Brazil

Question

How to reason about events and event handling code?

Results:

- In Ptolemy:
  - Greybox specifications, “translucid contracts” [Main conference]
- In other languages
  - This talk

Question for this talk:

How to extend translucid contracts to other languages?

Background + Our Previous Works

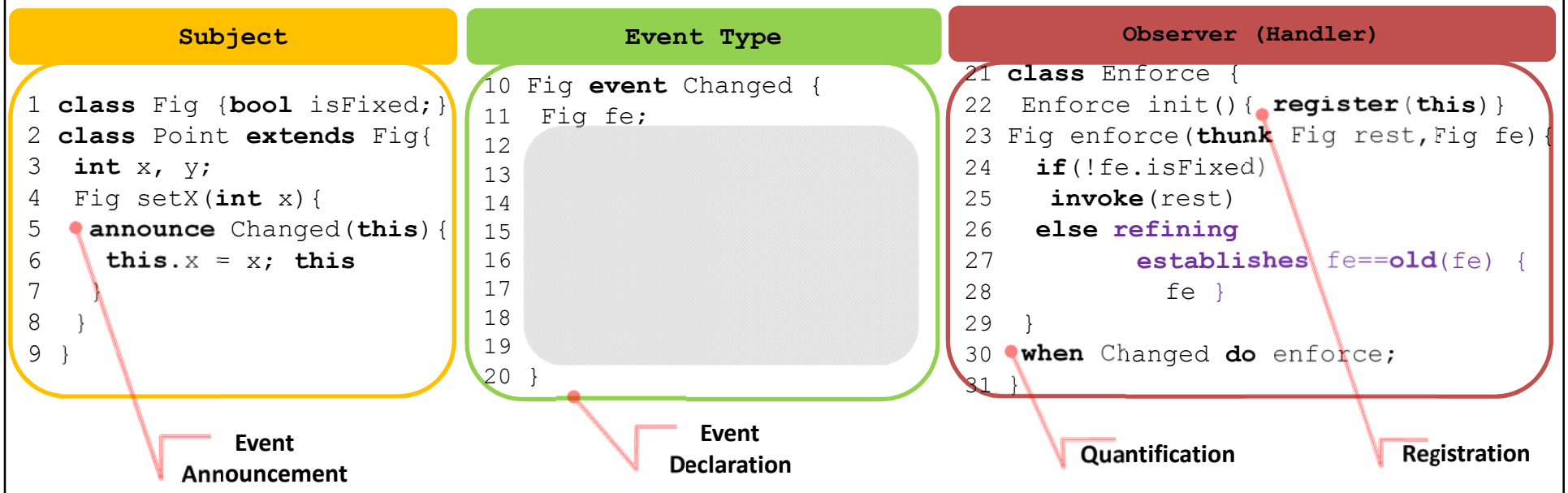
Problems for Modular Reasoning About control (with Advice)

1. Pervasive join point shadows:
   mySquare.setX (9);
2. Black-box specifications
   can't specify control effects of advice

Ptolemy Highlights

- Events:
  - Explicit Declaration
  - Explicit Announcement
  - Quantification for handlers
- Translucid Contracts

Ptolemy Example



- Skip execution of *setX()* when *isFixed* is true.
- Event-driven-programming:
  - **Subject** *Point* announces **event** *Changed* when *setX()* is called.
  - **Event handler** *enforce* registers for *Changed* and runs when the event is announced.
  - Handler *enforce* implements the above requirement.

Translucid Contracts Example

```
10 Fig event Changed {
11  Fig fe;
12  requires fe != null
13  assumes{
14   if(!fe.isFixed)
15    invoke(next)
16   else
17    establishes fe==old(fe)
18  }
19  ensures  fe != null
20 }
```

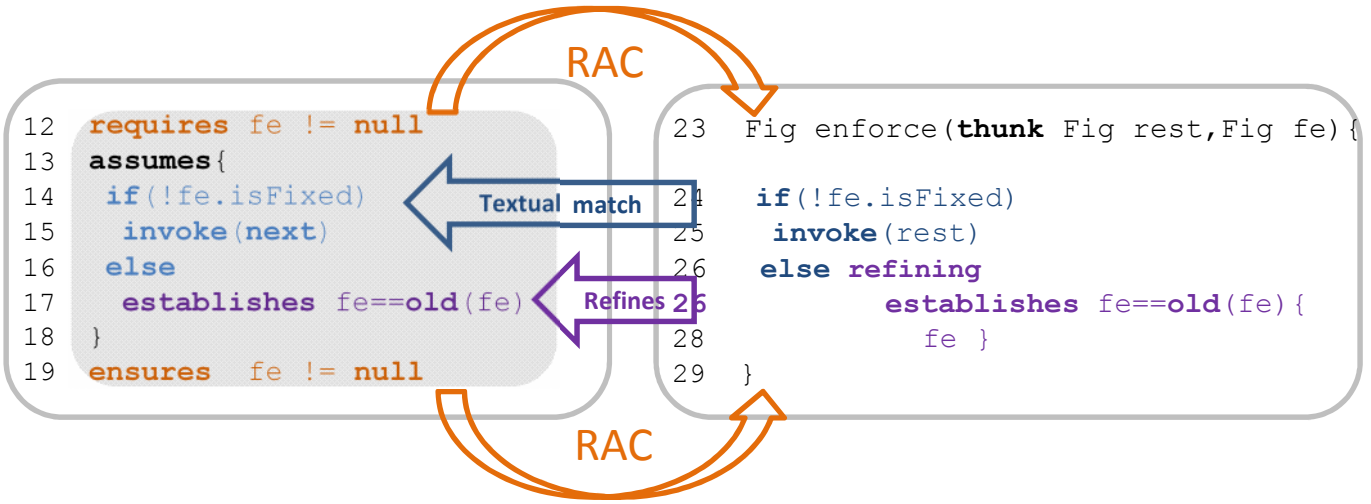
- Contract Limits the behavior of the handler
  - **requires/ensures** labels pre-/postconditions
- Greybox limits the handler's code
  - **assumes** block with program/spec. expressions

Assumes Block

```
10 Fig event Changed {
11  Fig fe;
12  requires fe != null
13  assumes{
14   if(!fe.isFixed)
15    invoke(next)
16   else
17    establishes fe==old(fe)
18  }
19  ensures  fe != null
20 }
```

- Assumes block is a mixture of
  - Specification exprs
    - Hide implementation details
  - Program exprs
    - Reveal code details
    - Must be present in code

TC Refinement



- A program expr. is refined by a **textually matching** prog. expr.
- A specification expr. is refined by a refining expression with the same specification.
- The pre-/postconditions are enforced using runtime assertion checks (RACs)
- Handler should **structurally match** the TC assumes block.

How Translucid Contracts rely on Ptolemy

```
21 class Enforce {
22  …
23  Fig enforce(thunk Fig rest,Fig fe){
24   if(!fe.isFixed)
25    invoke(rest)
26   else
27   refining establishes fe==old(fe){
28    fe }
29  }
30  when Changed do enforce;
31 }
```

Event Name

Handler Name

- Handlers statically know about types of event they handle.
- So can find contract for the handler, …
- Allows modular checking of handler's refinement.

Ptolemy and Modular Reasoning

- Reasoning about control effects of AO programs:
  1. Pervasive join point shadows
     1. Solution: Limit the application of advice:
        1. Ptolemy's explicit event announcement
  2. Black-box specifications can't handle control effects
     1. Solution: Use grey-box spec to specify control effects:
        1. Translucid contracts:
           1. Limit the behavior of advice

- Ptolemy +Translucid contracts enables modular reasoning about control effects

Observation

- Greybox specifications are not Ptolemy-specific
- We show applicability to:
  - Several AO interfaces
  - OO language, C#

Key Problem

- Statically find what types of events a handler can handle
- So can find specification for event handler

Applicability of Translucid Contracts to other AO interfaces

- AO interfaces make join points explicit
  - Mitigates problem of pervasive join point shadows.

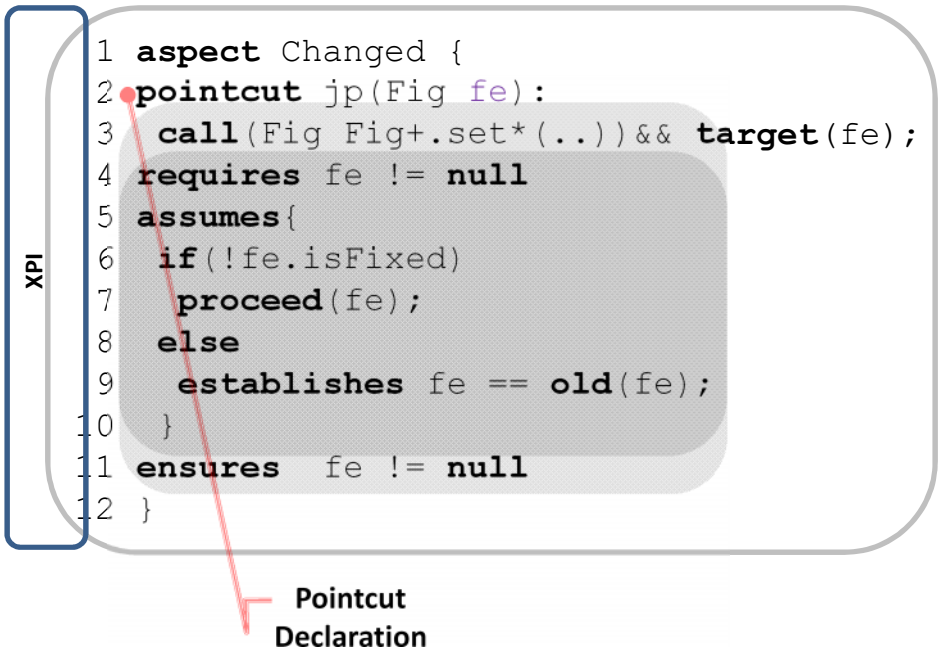
- Some AO interface proposals:
  - Crosscut programming interfaces (XPIs)
  - Aspect-aware interfaces (AAIs)
  - Open modules
  - Join point types
  - Explicit join points

Translucid Contracts for XPIs

- XPIs: a design-rule based interface; decouples the design of base and aspects

- XPI limits:
  - Exposure of join points
  - Behavior of base and aspect using black-box
    - No mechanism provided to check the refinement

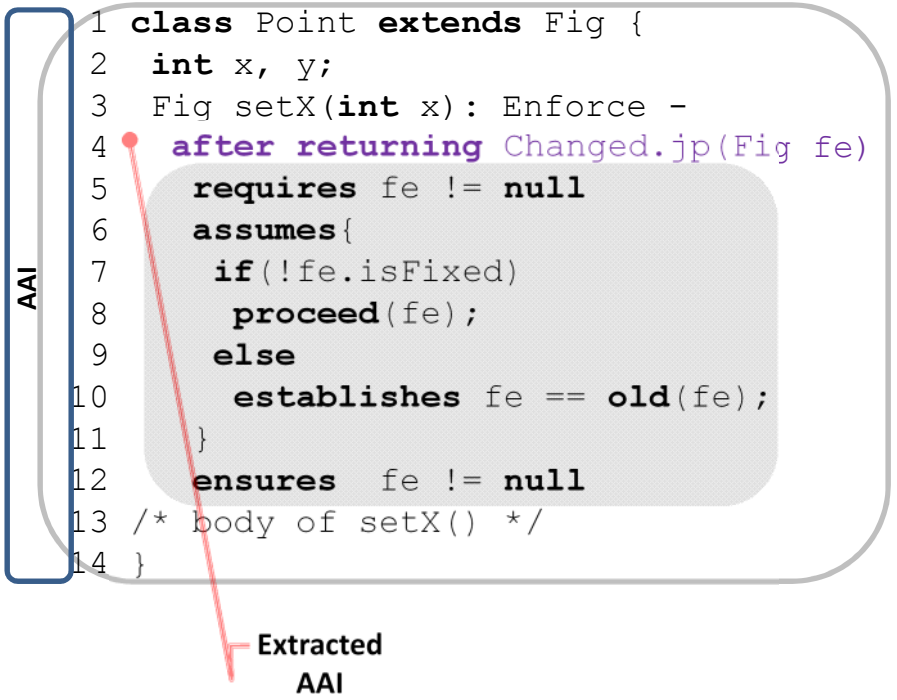
Translucid Contracts for XPIs




```
13 aspect Enforce {
14  Fig around(Fig fe): Changed.jp(fe){
15   if(!fe.isFixed)
16    proceed(fe);
17   else
18    refining establishes fe==old(fe){
19     return fe;
20   }
21  }
22 }
```

- Unlike Ptolemy, TC in XPI is attached to the pointcut rather than the event type
- Only variables in pointcut, *fe*, could be used in translucid contracts
- Refinement rules are added to AO type system in *Enforce* where it is reusing the pointcut *Changed.jp*

Translucid Contracts for AAIs



- Syntax/refinement rules are similar to XPI.

Translucid Contracts for Open Modules

- Open modules allow explicit exposure of pointcut for behavioral modifications , similar to signaling events in Ptolemy

Translucid Contracts for Open modules

Open Module

```
1  module Changed{
2   class Fig;
3   expose: XPI.jp(Fig)
4  }
5  aspect XPI {
6  pointcut jp(Fig fe):
7   call(Fig Fig+.set*(..))&& target(fe);
8  requires fe != null
9  assumes{
10  if(!fe.isFixed)
11   proceed(fe);
12  else
13   establishes fe == old(fe);
14  }
15 ensures  fe != null
16 }
```

Exposed Join Point

```
13 aspect Enforce {
14  Fig around(Fig fe): XPI.jp(fe)
15   if(!fe.isFixed)
16    proceed(fe);
17   else
18    refining establishes fe==old(fe){
19     return fe;
20   }
21  }
22 }
```

- Open module *changed* exposes a pointcut of class *Fig* to be advised by *Enforce.*
- Like XPIs and unlike Ptolemy, TCs are attached to pointcut decl.

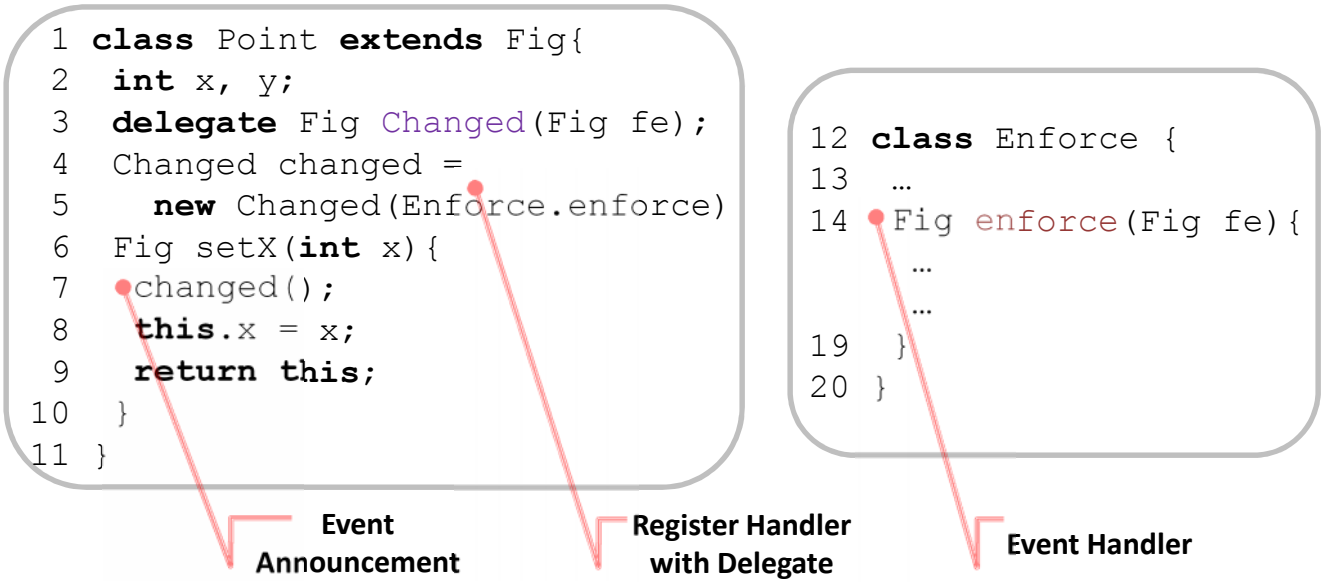
Translucid Contracts for Join Point Types & Explicit Join Points

- Join point types and Explicit join points are similar to Ptolemy's event types.

Applicability to OO languages

- For each handler, need to statically know about the type of events it can handle.
- C# has built-in:
  - events (EventType interface)
  - delegates (method pointers)

Problem Determining Handled Event Type of a Handler



- Any method with the same signature as delegate *Changed* could be registered as its handler.
- Handler *enforce* has no way to tell which event it is handling.

Solution:
A Simple Convention

- Every handler,
  takes as first argument an
  instance of the event type
  it handles

C# Library

- A C# library, emulating Ptolemy's functionality, using C# events and delegates developed

Event Declaration

C#

```
10 class Changed:EventType <Fig, Changed.Context>{
11  class Context{
12   Fig fe;
13   Context (Fig fe){ this.fe = fe;}
14   Fig contract() {
15    Contract.Requires(fe != null);
16    Contract.Ensures(fe != null);
17    if (!fe.isFixed)
18     return new Changed().Invoke();
19    else {
20     Contract.Assert(true);
21     Contract.Assert(fe==Contract.OldValue(fe));
22    }
23  }}}
```

Translucid Contract

Ptolemy

```
10 Fig event Changed {
11  Fig fe;
12  requires fe != null
13  assumes{
14   if(!fe.isFixed)
15    invoke(next)
16   else
17    establishes fe==old(fe)
18  }
19  ensures  fe != null
20 }
```

- Event *Changed* extends the built-in *EventType* with return type *Fig* and context variable *fe.*
- Translucid contract is attached to the event type *Changed*
- Method ***contract()*** documents the translucid contract.
- Pre/postconditions are written using **Code Contracts**
- Specification exprs are documented using assert exprs.

Event Announcement

C#

```
1  class Fig { int isFixed; }
2  class Point:Fig {
3   int x, y;
4   void setX(int x) {
5    Changed.Announce(
6       new Changed.Context(this),()=>{
7     this.x = x;
8     return this;});
9   }
10 }
```

Ptolemy

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

- Event body is provided as an anonymous closure.

Event Handler

C#

```
26 class Enforce {
27  Fig enforce(ChangedEvent next) {
28    if (!next.fe.isFixed)
29     return next.Invoke();
30    else {
31     return next.fe;
32    }
33  }
34 }
```

Refining Expression

Quantification

Ptolemy

```
21 class Enforce {
22  Enforce init(){ register(this)}
23  Fig enforce(thunk Fig rest,Fig fe){
24   if(!fe.isFixed)
25    invoke(rest)
26   else
27   refining establishes fe==old(fe){
28    fe }
29  }
30  when Changed do enforce;
31 }
```

- Passing the event type to the handler, emulates Ptolemy's quantification.
- Can statically determine which events a handler is handling.
- Then the event type contract could be pulled out and the handler refinement of the contract could be checked statically for structural refinement and dynamically by runtime assertion checking.

Conclusion

- Previous work: Translucid Contracts enable modular reasoning of control effects in AO programs. Implemented in Ptolemy
- This work: Translucid contracts are not Ptolemy-specific , can be used to reason about:
  - Other AO interfaces
  - OO languages such as C#

Questions