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

Mehdi Bagherzadeh
Gary T. Leavens
Robert Dyer
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   
13   requires fe != null
14   assumes{
15       if(!fe.isFixed)
16       invoke(next)
17   else
18       establishes fe==old(fe)
19   }
20   ensures fe != null
21 }
```

- 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.

---

```
requires fe != null
assumes {
  if (!fe.isFixed)
    invoke(next)
  else
    establishes fe == old(fe)
}
ensures fe != null

Fig enforce(thunk Fig rest, Fig fe) {
  if (!fe.isFixed)
    invoke(rest)
  else refining
    establishes fe == old(fe) {
      fe
    }
}
```
How Translucid Contracts rely on Ptolemy

- 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

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

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 AAls

- 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
• 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)
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

- 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

- Event body is provided as an anonymous closure.
Event Handler

- 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#