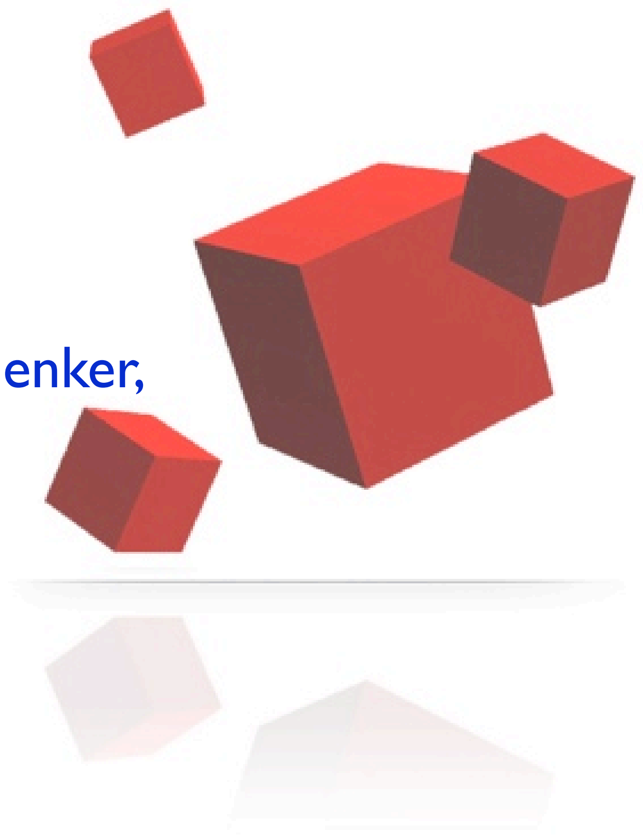
Context-Aware Aspects



Éric Tanter, Kris Gybels, Marcus Denker,
Alexandre Bergel
Trinity College, Dublin

`Alexandre.Bergel@cs.tcd.ie`

Vienna, March 26, 2006

Introduction

- Context awareness
  - program behavior depends on “context”
  - issue: if statements tangling
  - seen as a crosscutting concern
- Our approach: aspect language constructs
  - need for context abstractions in the language [R.Gabriel@aosd06]

Outline

1. Contexts with an Online Shopping Application
2. Context-Aware Aspects
3. Framework-based approach
4. Related work
5. Conclusion

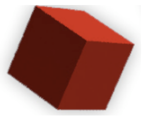
Contexts with an Online Shopping Application

When a purchase has to be ordered,
the bill is calculated.

```
aspect Discount {
 double rate = 0.90;

 pointcut amount():
   execution (double ShoppingCart.getAmount());

 double around():
   amount() {return proceed() * rate;}
}
```



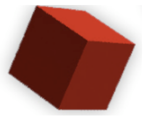
Variability in the relation Context-Aspect

- Discounting aspect can be based on
  - promotion when user **checks out**
  - promotion when user **logs in**
  - promotion when an item is **added to cart**
  - ...
- Promotional context can be based on
  - **time slots**
  - state of the **stock (overload)**
  - purchase done **via web service** (ie. control flow property)
  - ...
- Rate can be constant or **depend** on the promotion context

Separate contexts and aspects

Context: Part of the Environment

- *Stateful*: public and private data carried to describe an environment.
- *Composable*: elaborated contexts obtained from primitive contexts.
- *Parameterized*: generic context parametrized by aspects.



Restricting an Aspect to a Context - step 1

Reference to a context in the pointcut definition:

“apply discount if currently in promotion context”

```
aspect Discount {
 double rate = 0.90;
 pointcut amount():
   execution (double ShoppingCart.getAmount())
    && inContext(PromotionCtx);

 double around(): amount() {
  return proceed() * rate;
 }
}
```

context restriction

Restricting an Aspect to a Context - step II

Discount rate is determined by the context:

"... and accessing the rate"

```
aspect Discount {
 pointcut amount(double rate):
   execution (double ShoppingCart.getAmount())
    && inContext(PromotionCtx(rate));

 double around(double rate): amount(rate) {
   return proceed() * rate;
 }
}
```

context state exposure

Restricting an Aspect to a Context - step III

A context is parameterized by the dependent aspect:

"... if stock overload reaches 80%"

```
aspect Discount {
 pointcut amount(double rate):
   execution (double ShoppingCart.getAmount())
    && inContext(PromotionCtx(rate))
    && inContext(StockOverloadCtx[0.80]);

 double around(double rate): amount(rate) {
   return proceed() * rate;
 }
}
```

context parameterization

Extensible Context Restrictions

- General purpose restrictions:
  - **`inContext (c)`** : if current context = *c*
  - **`createdInContext (c)`**: if *this* was created in context *c*
  - **`...`**
- Domain/application-specific restrictions:
  - **`putInCartInContext (c)`**: if context when *this* was added to a cart = *c*
  - **`...`**

Extensible Context Restrictions

- General purpose restrictions:
  - **`inContext (c)`** : if current context = *c*
  - **`createdInContext (c)`**: if *this* was created in context *c*
  - **`...`**
- Domain/application-specific restrictions:
  - **`putInCartInContext (c)`**: if context when *this* was added to a cart = *c*
  - **`...`**

CONTEXT SNAPSHOTS

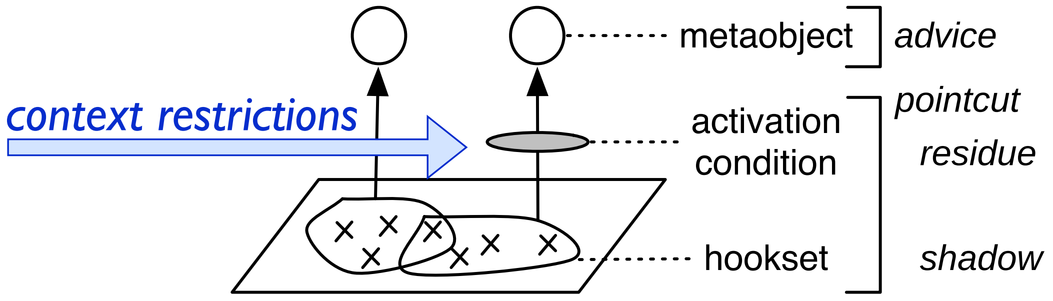
Context-Aware Aspects in a Nutshell

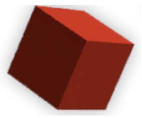
- Contexts and aspects are separated.
- Contexts are parameterized, composable and stateful.
- Context state bound to pointcut variables in aspects.
- Support for new context-related pointcut restrictors.

Implementation

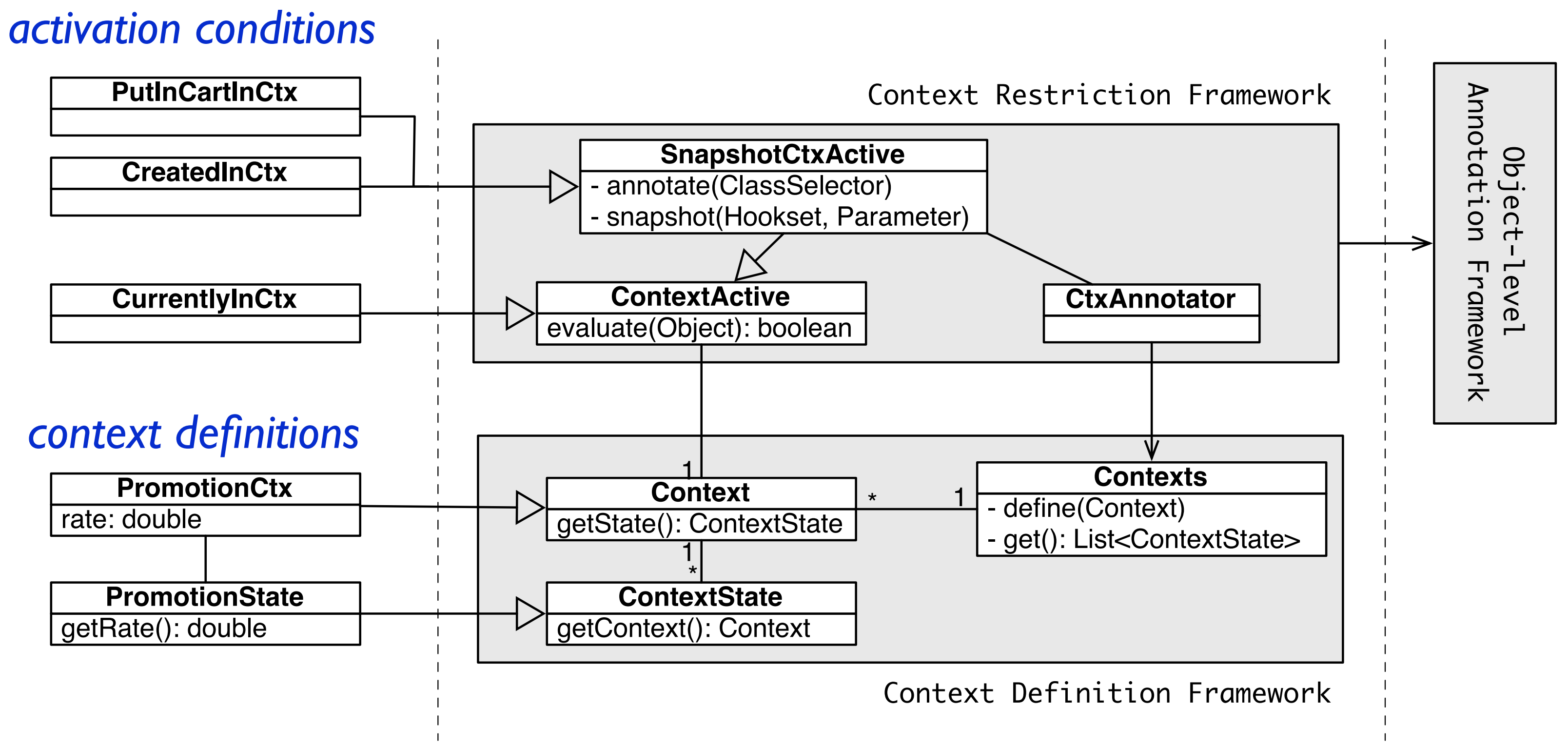
Implementation

- Requirements for an AOP framework (core semantics)
  - aspects first-class (eg. cflow exposed as an object)
  - extensibility of dynamic conditions
- Our implementation: Reflex
  - links as first-class pointcut/advice pairs
  - activation conditions as objects





Framework for Context-Aware Aspects



Definition of the Promotion context

. Promotion active when using web services (**control flow)**
. Reference to **past** context state: *which state to capture?*

```
class PromotionCtx implements Context {
  double rate = ...;  // variable state

  // cflow(execution(* WebServiceRequest+.*(..)))
  CFlow cf = CFlowFactory.get(
     new Hookset(MsgReceive.class, new NameCS("WebServiceRequest"),
                 new AnyOS()));

  ContextState getState() {
   return (cf.in())? new PromotionState(rate)
                   : null;
  }
}
```

Related Work

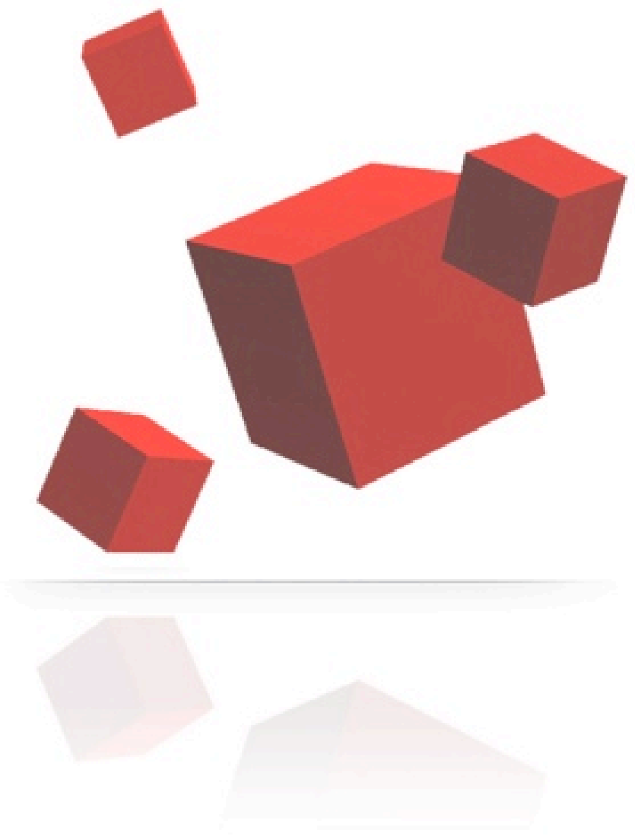
- ContextL [Dynamic Languages Symposium 2005]
  - language approach to context orientation, no aspects
- EAOP, stateful aspects, ...
  - focus on “internal context” (joinpoints), no notion of external ctx
- CaesarJ [TAOSD 2006]
  - thread-based scoping (kind of ctx)

Conclusion

- Proposed the notion of *context-aware aspects*
  - aspects that depend on context
  - new and extensible set of pointcut restrictors
- Framework for context-aware aspects based on Reflex.
- **Handling context-related behavior as aspects allows for a better modularization.**
- Future Work
  - Concrete syntax for context-aware aspects over Reflex
  - Apps in ubiquitous computing: eg. WildCAT for external context

Context-Aware Aspects

- Aspect behaviour depends on (possibly past) context
- Contexts
  - stateful
  - composable
  - parameterized
  - can be snapshot



Alexandre Bergel
Alexandre.Bergel@cs.tcd.ie