When Behavior Disappears

16 - 12 - 2013

Nicolás Cardozo

Vrije Universiteit Brussel (ncardozo@vub.ac.be)

Software↵ Languages.Lab

Context-oriented Programming

available contexts

system behavior

```
-(void) displayRoute {
  GMSCameraPosition *camera = ...
  GMSMapView *mapView = ...
  self.view = mapView;
}
```

context definitions

@context(Compass)

@context(GPS)

behavioral adaptations

Context-oriented Programming

available contexts

system behavior

```
-(void) displayRoute {
  GMSCameraPosition *camera = ...
  GMSMapView *mapView = ...
  self.view = mapView;
}
```

```
@contexts GPS
-(void) displayRoute {
  //set coordinates & zoom
  //display map
}
```

context definitions

@context(Compass)

@context(GPS)

behavioral adaptations

Context-oriented Programming

available contexts

system behavior

context definitions

```
@context(Compass)
@context(GPS)
```

Compass

GPS

```
-(void) displayRoute {
  GMSCameraPosition *camera = ...
  GMSMapView *mapView = ...
  self.view = mapView;
}
```

```
@contexts GPS
-(void) displayRoute {
  //set coordinates & zoom
  //display map
}
```

```
@contexts Compass
-(void) displayRoute {
  //set map rotation and inclination
  //display map
}
```

behavioral adaptations

Context-oriented Programming

available contexts

Compass

system behavior

```
-(void) displayRoute {
  GMSCameraPosition *camera = ...
  GMSMapView *mapView = ...
  self.view = mapView;
}
```

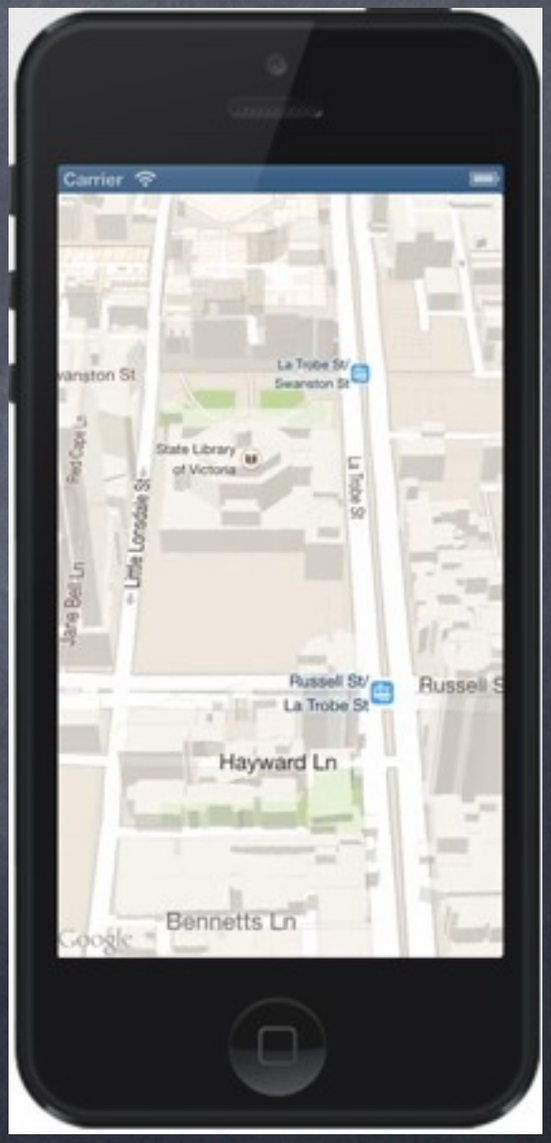
context definitions

```
@contexts Compass
-(void) displayRoute {
  //set map rotation and inclination
  //display map
}
```

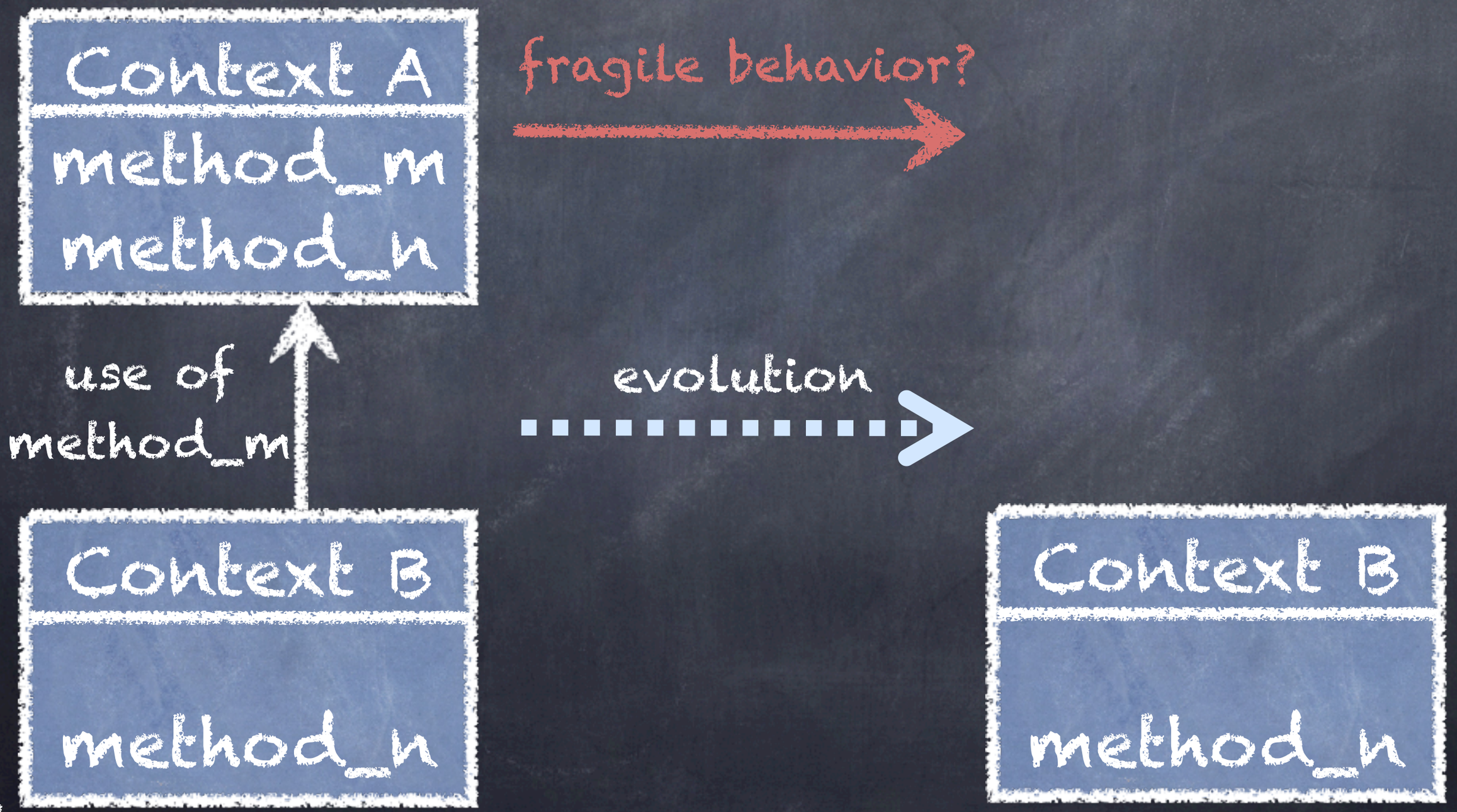
```
@context(Compass)
@context(GPS)
```

behavioral adaptations

Context-oriented Programming



The Fragile Method Problem



Fragility in Dynamically Adaptive Systems

- Software evolution without downtime.
- COP = base code + behavioral adaptations
- Behavioral adaptations = situation + behavior
- Software artifacts are composed dynamically from the available behavior provided by contexts.
- Fragility problems are thus a kind of behavior completeness and correctness problems

Dealing with fragility

```
@contexts GPS
-(void) displayRoute {
  location = [getLocation];
  GMSCameraPosition *camera =
      [GMSCameraPosition
        withLatitude:location.latitude
        longitude:location.longitude
        zoom: self.zoom];
  [self display:camera];
}

@context GPS
- (Location *) getLocation { ... }
```

```
@contexts Compass
-(void) displayRoute {
  GMSCameraPosition *camera =
      [GMSCameraPosition
        withLatitude:[getLocation].latitude
        longitude:[getLocation].longitude
        zoom: self.zoom
        heading.magnetic*Pi/180];
  [self display:camera];
  [mapView setTransform:
  CGAffineTransformMakeRotation(
                        -self.rotation)]; }
```

```
- void display: (GSMCameraPosition *) camera {
  GMSMapView *mapView = [GMSMapView mapWithFrame:CGRectZero
                                  camera:camera];

  GMSMarker *marker = [[GMSMarker alloc] init];
  marker.position = camera.target;
  self.view = mapView;  }
```

Dealing with fragility

```
@contexts GPS
-(void) displayRoute {
  location = [getLocation];
  GMSCameraPosition *camera =
      [GMSCameraPosition
        withLatitude:location.latitude
        longitude:location.longitude
        zoom: self.zoom];
  [self display:camera];
}

@context GPS
- (Location *) getLocation { ... }
```

assumed dependency

```
@contexts Compass
-(void) displayRoute {
  GMSCameraPosition *camera =
      [GMSCameraPosition
        withLatitude:[getLocation].latitude
        longitude:[getLocation].longitude
        zoom: self.zoom
        heading.magnetic*Pi/180];
  [self display:camera];
  [mapView setTransform:
  CGAffineTransformMakeRotation(
                        -self.rotation)]; }
```

```
- void display: (GSMCameraPosition *) camera {
  GMSMapView *mapView = [GMSMapView mapWithFrame:CGRectZero
                                          camera:camera];

  GMSMarker *marker = [[GMSMarker alloc] init];
  marker.position = camera.target;
  self.view = mapView;  }
```

Dealing with fragility

Program execution

program instructions	external events	active contexts
⋮	@activate(GPS)	{GPS}
	@activate(Compass)	{Compass, GPS}
[displayRoute]@Compass		
[getLocation]@GPS		
[display: camera]@Default		
⋮		

Dealing with fragility

Program execution

program instructions	external events	active contexts
⋮	@activate(GPS)	{GPS}
	@activate(Compass)	{Compass, GPS}
[displayRoute]@Compass		
	@deactivate(GPS)	{Compass}
[getLocation]@GPS		
[display: camera]@Default		
⋮		

Dealing with fragility

program instructions	external events	active contexts
⋮	@activate(GPS)	{GPS}
	@activate(Compass)	{Compass, GPS}
[displayRoute]@Compass		
	@deactivate(GPS)	{Compass}
[getLocation]@GPS		
[display: camera]@Default		
⋮		

MessageNotUnderstood
method should be
available as long as
Compass context is active

Program execution

Dealing with fragility

displayRoute

getLocation

GPS

Default

Compass

Default

GPS

Default

method call

specialization

accessible from

Dealing with fragility

@deactivate(GPS)

{-getLocation}

displayRoute

getLocation

required specialization of getLocation in the Compass context

Default

Compass

method call

specialization

accessible from

Resolution

Reuse contracts

$$\text{Compass} \Rightarrow \exists c(@\text{Context})\ @\text{active}(c) \wedge c \longrightarrow \text{getLocation}$$

- Reuse contracts define a contract specifying the assumptions of the user about the "base" behavior
- Unidirectional contracts
- Defines who can provide the desired behavior
- Evolution operator given by snapshots of the composed system

Reuse contracts

displayRoute

getLocation

{compass, GPS}

{GPS}

GPS

Default

Default

GPS

Compass

method call

specialization

accessible from

Reuse contracts

@deactivate(GPS)

{-getLocation}

displayRoute

getLocation

{compass, GPS}

{GPS}

GPS

Default

Compass

method call

specialization

accessible from

Reuse contracts

@deactivate(GPS)

{-getLocation}

displayRoute

getLocation

{compass, GPS}

{GPS}

GPS

Default

Compass

method call

specialization

accessible from

Reuse contracts

@deactivate(GPS)

{-getLocation}

displayRoute

getLocation

{compass, GPS}

{GPS}

GPS

Default

Compass

method call

specialization

accessible from

Reuse contracts

	+X.getLocation	-X.getLocation	getLocation refined	getLocation generalized	getLocation changed	...
display{getLocation}	✓	method does not exist	✓	missed assumed behavior	✓	...
...	...	...	...	...	...	...

Identification

Reuse contracts

Compass $\Rightarrow$ $\exists$c(@Context) @active(c)

$\wedge$ c $\longrightarrow$ getLocation

Context dependency relations

Implication

[GPS addImplicationTo: Compass]

Causality

Requirement

[Cardozo et al. Context-oriented programming 2011]

[Cardozo et al. Petri nets and Software Engineering 2012]

Identification

Runtime

- Direct completeness verification
- Only default resolution strategies

Upfront analysis

- Generation of possible program states
- Ensure completeness

- State explosion
- Uncertainty about incoming adaptations

Conclusion

- Dynamic adaptation is nice, but unreliable
- Ideas from static settings can be reused in a dynamic context
  - Special attention to identification and resolution
- Tradeoff between completeness, consistency and promptness