Chaperones and Impersonators

Run-time Support for Reasonable Interposition

T. Stephen Strickland
Univ. of Maryland, College Park

Sam Tobin-Hochstadt
Northeastern Univ.

Robert Bruce Findler
Northwestern Univ.

Matthew Flatt
Univ. of Utah

OOPSLA, October 25, 2012
Contract Systems
Recent Uses of Contract Systems

Static Analysis


Leino. Staged Program Development. OOPSLA 2012 Keynote.


Type Systems

Chugh et al. Dependent Types for JavaScript. OOPSLA 2012.

Takikawa et al. Gradual Typing for First-Class Classes. OOPSLA 2012.
Higher-order Contracts

Contracts specified separately from values
Specifications may describe higher-order behavior
Contract system provides blame tracking

server

client

(→ prime? string?)
Higher-order Contracts

Contracts specified separately from values
Specifications may describe higher-order behavior
Contract system provides blame tracking

server

client

\[ \lambda \]

\[ \lambda \]

\((\rightarrow \text{prime?} \; \text{string?})\)

client broke the contract, expected prime?, got 4
Higher-order Contracts

Contracts specified separately from values

Specifications may describe higher-order behavior

Contract system provides blame tracking

server

\[ \lambda \]

\( \Rightarrow \) prime? \textbf{string?} \)

client

\[ \lambda \]

\( 5 \)

\( 8 \)

server broke the contract, expected \textbf{string?}, got 8
Prior Support for Higher-order Contracts

Functions

Immutable containers ✔

Mutable containers ✗

Generative structures ✗
Prior Support for Higher-order Contracts

- Functions ✓
- Immutable containers ✓
- Mutable containers X
- Generative structures X

Let's use proxies!
The Problem with Proxies
The Problem with Proxies
The Problem with Proxies
The Problem with Proxies
Our System of Proxies

Chaperones

Restricted in changing behavior
Applicable to more values

Impersonators

Freer to change behavior
Applicable to fewer values
Current Support for Higher-order Contracts

- Functions ✅
- Immutable containers ✅
- Mutable containers ✅
- Generative structures ✅
Vector Chaperones

\[(\text{define } \text{vec}_1 \ (\text{vector } 2 \ 3 \ 5 \ 7))\]
Vector Chaperones

(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
))
Vector Chaperones

(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
    ; Interpose for vector-ref
    (λ (vec i v) (contract prime? v svv clt)))
  ))
Vector Chaperones

(define vec₁ (vector 2 3 5 7))
(define vec₂
  (chaperone-vector vec₁
    ; Interpose for vector-ref
    (λ (vec i v) (contract prime? v srv clt))
    ; Interpose for vector-set!
    (λ (vec i v) (contract prime? v clt srv)))))

clt broke its contract, expected prime?, got 8
Vector Chaperones

```
(define vec_1 (vector 2 3 5 7))
(define vec_2
  (chaperone-vector vec_1
    ; Interpose for vector-ref
    (λ (vec i v) 10)
    ; Interpose for vector-set!
    (λ (vec i v) (contract prime? v clt srv))))
```

non-chaperone result, original: 5, received: 10
Chaperone Restriction

Results of interposition functions must be a chaperone of the appropriate input.

\[
(chaperone\text{-}of? \ v_1 \ v_2)
\]

- If \( v_1 \) is equal to \( v_2 \), true.
- If \( v_1 \) is a chaperone of \( v_3 \), then check \( (chaperone\text{-}of? \ v_3 \ v_2) \).
- Otherwise, false.
(struct fish (name weight))
(define f₁ (fish "Dory" 14))
(define f₂
  (chaperone-struct f₁
                  ; Operation to interpose
                fish-weight
                  ; Interposing function
                (λ (s v) (contract prime? v srv clt))
                ...))
Structure Chaperones

(struct fish (name weight))
(define f_1 (fish "Dory" 14))
(define f_2
  (chaperone-struct f_1
    ; Operation to interpose
    fish-weight
    ; Interposing function
    (λ (s v) (contract prime? v srv clt))
    ...))

srv broke its contract, expected prime?, got 14
Chaperone Limitations

Inputs and results of operations must behave like originals.

Sealing contracts


Takikawa et al.  Gradual Typing for First-Class Classes.  OOPSLA 2012.
Chaperone Limitations

\[
\text{vector-map: } (\forall/c \ [A \ B] \ ((A \ . \ -> \ . \ B) \\
\text{ (Vectorof } A) \ . \ -> \ . \ \\
\text{(Vectorof } B))))
\]
Chaperone Limitations

\[ \text{vector-map: } (\forall c \ [A \ B] \ ((A \ . \rightarrow \ B) \ (\text{Vectorof } A) \ . \rightarrow \ B) \ (\text{Vectorof } B))) \]
Vector Impersonators

```
(define vec₁ (vector 2 3 5 7))
(define vec₂
  (impersonate-vector vec₁
    ; Interpose for vector-ref
    (λ (vec i v) (contract prime? v srv clt)))
  ; Interpose for vector-set!
  (λ (vec i v) (contract prime? v clt srv))))
```

![Diagram of vector and its elements]
Vector Impersonators

(define vec₁ (vector 2 3 5 7))
(define vec₂
  (impersonate-vector vec₁
    ; Interpose for vector-ref
    (λ (vec i v) 10)
    ; Interpose for vector-set!
    (λ (vec i v) (contract prime? v clt srv))))
Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

```
(struct fish (name weight))
(define f₁ (fish "Dory" 12))
(define f₂
  (let ([counter (fish-weight f₁)])
    (impersonate-struct f₁
      fish-weight
      (λ (f v)
        (begin0 counter (set! counter 1)))))
```
Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

(struct fish (name weight))
(define f₁ (fish "Dory" 12))
(define f₂
  (let ([counter (fish-weight f₁)])
    (impersonate-struct f₁
      fish-weight
      (λ (f v)
        (begin0 counter (set! counter 1)))))

![Image of scales with fish on each scale]
Impersonator Restrictions

No impersonators for immutable containers.

No impersonators for immutable fields of generative structures.

(struct fish (name weight))
(define f1 (fish "Dory" 12))
(define f2
  (let ([counter (fish-weight f1)])
    (impersonate-struct f1
      fish-weight
      (λ (r v)
        (begin0 counter (set! counter 1)))))))
Contract Hierarchy

Flat $\subset$ Higher Order

Flat $\subset$ Chaperones $\subset$ Impersonators

eq? chaperone-of? No restriction


A General Proxying System

Revocable Membranes


Local views on remote data

SMTP server access that appears like local hash tables and vectors
## Performance (in seconds)

<table>
<thead>
<tr>
<th>benchmark</th>
<th>no proxy</th>
<th>no checks</th>
<th>impersonate</th>
<th>chaperone</th>
</tr>
</thead>
<tbody>
<tr>
<td>make guide</td>
<td>10.467</td>
<td>10.606</td>
<td>10.818</td>
<td>10.792</td>
</tr>
<tr>
<td>render guide</td>
<td>1.889</td>
<td>2.044</td>
<td>3.741</td>
<td>3.727</td>
</tr>
<tr>
<td>keyboard</td>
<td>5.182</td>
<td>5.231</td>
<td>7.253</td>
<td>7.258</td>
</tr>
<tr>
<td>slideshow</td>
<td>4.663</td>
<td>4.776</td>
<td>5.168</td>
<td>5.180</td>
</tr>
<tr>
<td>plot</td>
<td>1.854</td>
<td>1.886</td>
<td>2.362</td>
<td>2.394</td>
</tr>
<tr>
<td>typecheck</td>
<td>22.610</td>
<td>24.144</td>
<td>47.302</td>
<td>47.816</td>
</tr>
<tr>
<td>ode-apply</td>
<td>7.794</td>
<td>9.265</td>
<td>10.236</td>
<td>10.632</td>
</tr>
</tbody>
</table>
## Performance (in seconds)

<table>
<thead>
<tr>
<th>benchmark</th>
<th>no proxy</th>
<th>no checks</th>
<th>impersonate</th>
<th>chaperone</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>make guide</td>
<td>10.467</td>
<td>10.606</td>
<td>10.818</td>
<td>10.792</td>
<td>0%</td>
</tr>
<tr>
<td>render guide</td>
<td>1.889</td>
<td>2.044</td>
<td>3.741</td>
<td>3.727</td>
<td>0%</td>
</tr>
<tr>
<td>keyboard</td>
<td>5.182</td>
<td>5.231</td>
<td>7.253</td>
<td>7.258</td>
<td>0%</td>
</tr>
<tr>
<td>slideshow</td>
<td>4.663</td>
<td>4.776</td>
<td>5.168</td>
<td>5.180</td>
<td>0%</td>
</tr>
<tr>
<td>plot</td>
<td>1.854</td>
<td>1.886</td>
<td>2.362</td>
<td>2.394</td>
<td>1%</td>
</tr>
<tr>
<td>typecheck</td>
<td>22.610</td>
<td>24.144</td>
<td>47.302</td>
<td>47.816</td>
<td>1%</td>
</tr>
<tr>
<td>ode-apply</td>
<td>7.794</td>
<td>9.265</td>
<td>10.236</td>
<td>10.632</td>
<td>4%</td>
</tr>
</tbody>
</table>
Conclusion

Unrestricted proxies break programmer and compiler invariants.

Providing restricted proxies avoids this issue.

We now provide both via chaperones and impersonators in Racket.

http://racket-lang.org/