iSAQB Advanced DSL - Effects

Michael Sperber

Created: 2025-07-21 Mon 10:41

Describing Surface Analysis Processes

Requirements for a Lspecs DSL

- long-running processes
- variables
- simple arithmetic
- commands with arguments
- simple exception handling: abort current sequence of commands, run exception handler instead
- loops over static sets of numbers
- execution needs to keep log of commands run
- intermediate state of execution needs to be saved
- should always terminate

(LG 3-2) Principle of Least Power

The least expressive computation model should be chosen to represent the business domain.

(LG 3-2, LG 3-3) Exercise: Lspecs Design

- Design an Lspecs DSL!
- What would be an appriopriate formalism for describing the semantics?
- Describe the semantics?
- How expressive is the resulting DSL?
- What parts of an Lspecs program are static, which ones are dynamic?

Exercise: Denotation for Actions?

An action:

- needs to cause a log entry
- can either produce a valid result or report an exception

How can we represent this denotationally?

Monad

```
(: unit (%a -> (m %a))

(: bind ((m %a) (%a -> (m %b)) -> (m %b)))
```

Monads in the Wild

```
interface Stream<T> {
    static <T> Stream<T> of(T... values);
    <R> Stream<R> flatMap
    (Function<? super T,? extends Stream<? extends R>> mapper);
}
```

Monads in the Wild

```
interface Stream<T> {
    static <T> Stream<T> of(T... values);
    <R> Stream<R> flatMap
    (Function<T,  Stream<R>> mapper);
}
```

```
(: unit (%a -> (m %a))
```

```
(: bind ((m %a) (%a -> (m %b)) -> (m %b)))
```

```
interface Optional<T> {
  static <T> Optional<T> of(T value);
  <U> Optional<U> flatMap
  (Function<T, Optional<U>> mapper);
}
```

Exercise: Monad for Lspecs

Implement `unit` and `bind` operations for Lspecs!

Exercise: Embedded Syntax for Lspecs

Implement a more convenient syntax for Lspecs using macros!

(LG 4-4) Programs as Data

(See ../specs/specs-free-monad.rkt.)

Continuation: Unary function that continues execution after the "current" computation is done.

(LG 4-4) Common Effects

- Log is a *writer* effect: Items get added, don't get inspected by the Lspecs program.
- Exceptions are a *control* effect - they affect computation in a non-local fashion.

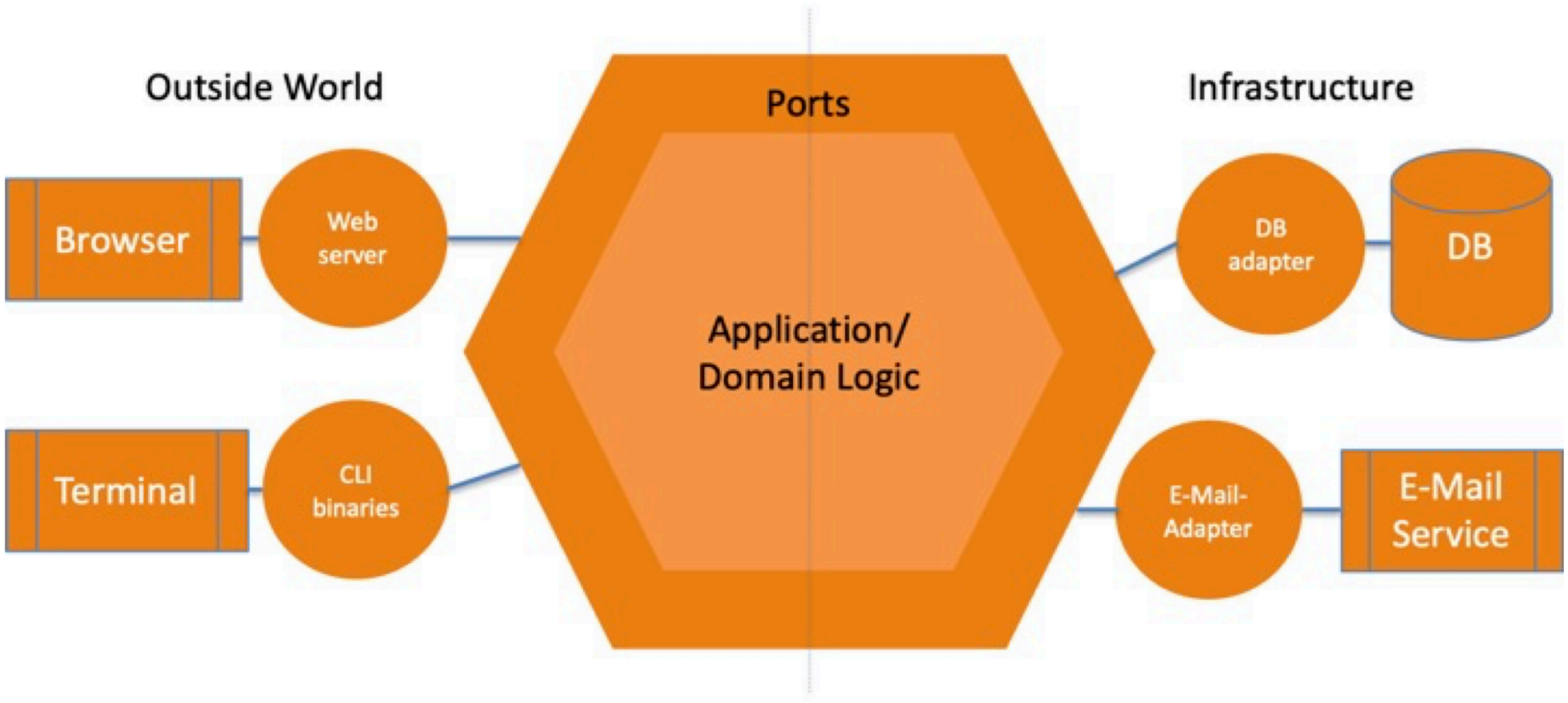
(LG 4-4) Other Effects

- Reader effect: has an `ask` operation for querying a value from the environment (e.g. Unix environment)
- State effect: has both `get` and `put` operations (essentially a mutable variale)
- Any kind of I/O

(LG 4-4) Dependency Injection / Adapter

```
(define (run comp handler-comp log)
  (cond
    ((unitm? comp)
     (values (unitm-result comp) log))
    ((failm? comp)
     (values exception-singleton log))
    ((handlem? comp)
     (let-values (((result log)
                   (run (handlem-computation comp)
                        (handlem-handler-computation comp) log)))
       (run ((handlem-continuation comp) result) handler-comp log)))
    ((commandm? comp)
     (values #f
             (cons (cons (commandm-text comp) (commandm-arg comp))
                   log)))))
```

(LG 4-4) Adapters as Effects



Exercise: Reduction Semantics

Implement a PLT Redex semantics for Lspecs!

(With help!)