Effects in elm

Presentation of commands, subscriptions and tasks

Definitions

Effect:

A change which is a result or consequence of an action or other cause.

Definitions

Side effect:

A secondary, typically undesirable effect of a drug or medical treatment.

Definitions

Effet de bord (informatique):

Une fonction est dite à effet de bord si elle modifie un état en dehors de son environnement local, c'est-à-dire a une interaction observable avec le monde extérieur autre que retourner une valeur.

Definitions

Effet de bord (informatique):

Une fonction est dite à effet de bord si elle modifie un état en dehors de son environnement local, c'est-à-dire a une interaction observable avec le monde extérieur autre que retourner une valeur.

→ le temps (Time), l'aléatoire (Random), les événements du DOM, les requêtes HTTP, la communication avec JavaScript (Port), ...

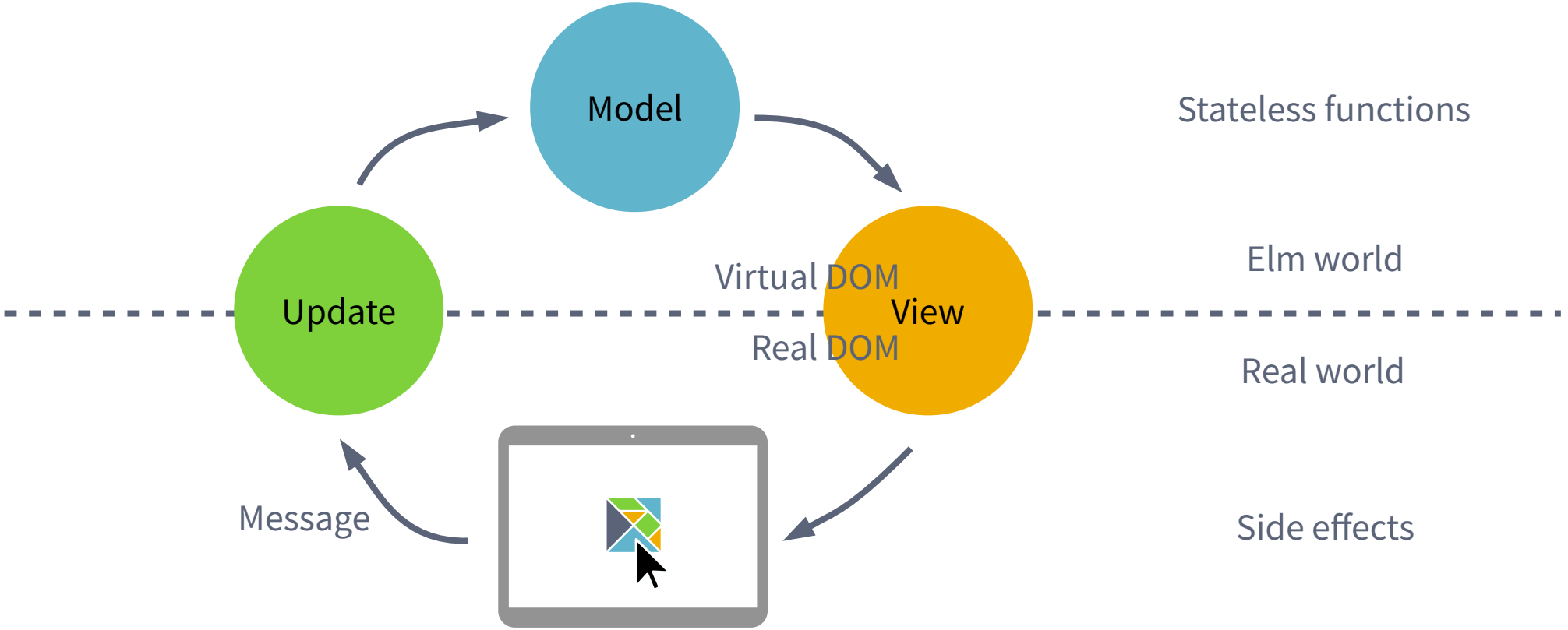
*** TEA – The Elm Architecture ***

Commands

Subscriptions

Tasks

Model – View – Update – unidirectional architecture



The elm Runtime

```
Browser.element :
    { init : flags -> ( model, Cmd msg )
    , view : model -> Html msg
    , update : msg -> model -> ( model, Cmd msg )
    , subscriptions : model -> Sub msg
    }
    -> Program flags model msg
```

The elm Runtime

```
Browser.element :
    { init : ...
    , view : ...
    , update : msg -> model -> ( model, Cmd msg )
    , subscriptions : ...
    }
    -> Program ...
```

The elm Runtime

```
Browser.element :
    { init : ...
    , view : ...
    , update : ...
    , subscriptions : model -> Sub msg
    }
    -> Program ...
```

TEA – The Elm Architecture

*** **Commands** ***

Subscriptions

Tasks

Http – get

```
Http.get : { url : String, expect : Expect msg } -> Cmd msg


-- Logic for interpreting a response body.
type Expect msg


Http.expectString : (Result Error String -> msg) -> Expect msg
```

Http – get

https://ellie-app.com/4M9VHByvkLMa1

```elm
type Msg
    = GotText (Result Http.Error String)


Http.get
    { url = "https://swapi.co/api/"
    , expect = Http.expectString GotText
    }
```

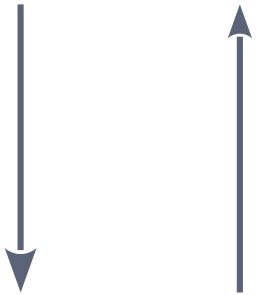
Http – post

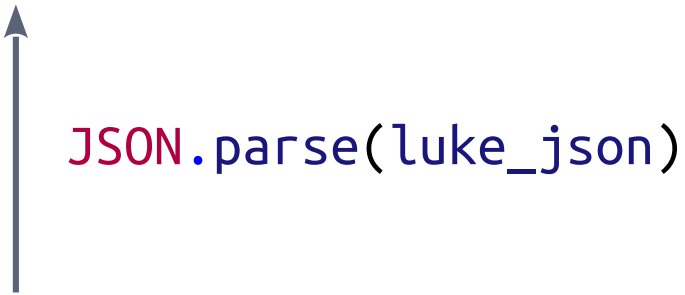
```
Http.post :
    { url : String
    , body : Body
    , expect : Expect msg
    }
    -> Cmd msg


-- Represents the body of a Request.
type Body


Http.fileBody : File -> Body
```

JSON – JavaScript Object Notation

JavaScript

```
luke = { name: "Luke Skywalker", height: 172 }
```






```
luke_json = '{"name":"Luke Skywalker","height":172}'
```

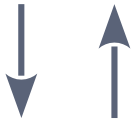
JSON

JSON – Encoding

elm

```
Encode.string : String -> Value
Encode.int : Int -> Value
Encode.bool : Bool -> Value
Encode.list : (a -> Value) -> List a -> Value
...
```

JavaScript `Value`



JSON

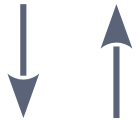
JSON – Encoding

elm

```
Encode.string : String -> Value
Encode.int : Int -> Value
Encode.bool : Bool -> Value
Encode.list : (a -> Value) -> List a -> Value
...
```

JavaScript `Value`

```
Encode.encode : Int -> Value -> String
```



JSON

JSON – Encoding

elm

```
Encode.string : String -> Value
Encode.int : Int -> Value
Encode.bool : Bool -> Value
Encode.list : (a -> Value) -> List a -> Value
...
```

?

JavaScript `Value`

```
Encode.encode : Int -> Value -> String
```

?

JSON

JSON – Decoding

```elm
-- Run a Decoder on some JSON Value.
Decode.decodeValue : Decoder a -> Value -> Result Error a


-- Parse the given string into a JSON value
-- and then run the Decoder on it.
Decode.decodeString : Decoder a -> String -> Result Error a
```

JSON – Decoding

```elm
-- Primitives

Decode.string : Decoder String
Decode.bool : Decoder Bool
...

-- Structures

Decode.list : Decoder a -> Decoder (List a)
Decode.dict : Decoder a -> Decoder (Dict String a)
Decode.field : String -> Decoder a -> Decoder a
...

-- Composition
Decode.map : (a -> b) -> Decoder a -> Decoder b
Decode.map2 : (a -> b -> c) -> Decoder a -> Decoder b -> Decoder c
...
```

JSON – Decoding

```elm
type alias People =
    { name : String
    , height : Int
    }


peopleDecoder : Decoder People
peopleDecoder =
    Decode.map2 People
        (Decode.field "name" Decode.string)
        (Decode.field "height" Decode.int)


Decode.decodeString peopleDecoder
    """{"name":"Luke Skywalker","height":172}"""
-- Ok { name = "Luke Skywalker", height = 172 }
```

JSON – Decoding

https://ellie-app.com/4McqN9PYSvpa1

Random

```elm
-- A Generator is a recipe for generating random values.
type Generator a

-- Create a command that produces random values.
Random.generate : (a -> msg) -> Generator a -> Cmd msg

-- Primitives
Random.int : Int -> Int -> Generator Int
Random.float : Float -> Float -> Generator Float
...

-- Data Structures
Random.pair : pair : Generator a -> Generator b -> Generator ( a, b )
...

-- Composition
Random.map : (a -> b) -> Generator a -> Generator b
...
```

TEA – The Elm Architecture

Commands

*** Subscriptions ***

Tasks

Subscriptions

A command : 1 “question” → 1 “answer”
A subscription : 1 “question” → n “answers”

```
Browser.element :
    { init : ...
    , view : ...
    , update : ...
    , subscriptions : model -> Sub msg
    }
    -> Program ...
```

Subscriptions

```elm
-- Get the current time periodically (interval in ms).
Time.every : Float -> (Posix -> msg) -> Sub msg

-- An animation frame triggers about 60 times per second.
-- Get the POSIX time on each frame.
Browser.Events.onAnimationFrame : (Posix -> msg) -> Sub msg

-- Subscribe to get codes whenever a key goes down.
Browser.Events.onKeyDown : Decoder msg -> Sub msg

-- Subscribe to any changes in window size.
Browser.Events.onResize : (Int -> Int -> msg) -> Sub msg
```

TEA – The Elm Architecture

Commands

Subscriptions

*** Tasks ***

Tasks

Tasks “describe asynchronous operations that may fail”.

```
type alias Task x a

type Result x a = Ok a | Err x


-- A task that succeeds immediately when run.
Task.succeed : a -> Task x a

-- A task that fails immediately when run.
Task.fail : x -> Task x a
```

Tasks – performing a task

We need to ask the elm runtime to “perform” a task and let us know when a result is returned.

```
-- Ask the runtime to perform a task that cannot fail,
-- like asking the current time:
-- Time.now : Task x Posix
Task.perform : (a -> msg) -> Task Never a -> Cmd msg


-- Similar to perform, but for a task that mail fail.
-- Thus the reason for the Result to handle.
Task.attempt : (Result x a -> msg) -> Task x a -> Cmd msg
```

Tasks – composing tasks

```elm
-- Chain together a task and a callback.
Task.andThen : (a -> Task x b) -> Task x a -> Task x b

timeInOneHour : Task x Time.Posix
timeInOneHour =
    Process.sleep (60 * 60 * 1000)
        |> Task.andThen (\_ -> Time.now)



-- Put the results of two tasks together.
Task.map2 : (a -> b -> c) -> Task x a -> Task x b -> Task x c
...
```

Difference between Task and Command

From faq.elm-community.org:
http://faq.elm-community.org/#what-is-the-difference-between-cmd-and-task

Cmd is just a bag (i.e. multiset) of chunks of data.
It is a functor, but it is not applicative or monadic.
This means all you can do is apply a function to all the entries
in the bag with map and add to the bag with batch.

Task is a way doing things in sequence.
It is monadic, meaning it has an andThen in the API.

Difference between Task and Command

From slack discussions:
https://gist.github.com/alpacaaa/133352462340423958l3d97af029b10f

evancz Mar 23, 2017 00:43
Just so folks are aware, one of the hard things about having ports just be a Task is the following.
Right now, a Task is guaranteed to terminate with an error or a result.

Questions ?

Exercice

Après une durée aléatoire, comprise entre 3 et 5 secondes, récupérer en une seule commande, le nom de la planète d'origine de la première personne de l'API de Star Wars.

Point de départ :
https://ellie-app.com/4Mf25C8bhN5a1