Domain Specific Languages
2: Building DSLs

Greg Michaelson
School of Mathematical & Computer Sciences
Heriot Watt University
Pragmatic approach

• usually nothing like formal language design
• start with basic idea of what DSL is for
• implement:
  – data structures
  – functions
• invent concrete syntax
• bridge concrete syntax to implementation
Pragmatic approach

• unsystematic
• over focus on concrete syntax
• hard to change mind
  – accretes features
  – spend time working round earlier decisions
• error prone
Systematic approach

1. functionality
   - underlying behaviours $\rightarrow$ representations/library

2. abstract syntax
   - meaningful constructs $\rightarrow$ abstract data type

3. semantics
   - map abstract syntax maps functionality $\rightarrow$
     interpreter calling library

4. concrete syntax
   - what user sees $\rightarrow$ parser
Example: functionality

• fish and chip shop
  - chips: £2.50
  - haddock: £6.50
  - cod: £5.45
  - pie: £2.75
  - fish cake: £1.20
  - chicken: £5.95
  - sausage: £1.25
  - black pudding: £2.90
  - haggis: £3.10

• menu: item -> price
Example: functionality

• fish and chip shop

  - chips £2.50
  - haddock £6.50
  - cod £5.45
  - pie £2.75
  - fish cake £1.20
  - chicken £5.95
  - sausage £1.25
  - black pudding £2.90
  - haggis £3.10

• menu: item -> price

• shop operations:
  - add: (item -> price) -> menu -> menu
  - delete: item -> menu -> menu
  - change: (item -> price) -> menu -> menu
Example: functionality

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>chips</td>
<td>£2.50</td>
</tr>
<tr>
<td>haddock</td>
<td>£6.50</td>
</tr>
<tr>
<td>cod</td>
<td>£5.45</td>
</tr>
<tr>
<td>pie</td>
<td>£2.75</td>
</tr>
<tr>
<td>fish cake</td>
<td>£1.20</td>
</tr>
<tr>
<td>chicken</td>
<td>£5.95</td>
</tr>
<tr>
<td>sausage</td>
<td>£1.25</td>
</tr>
<tr>
<td>black pudding</td>
<td>£2.90</td>
</tr>
<tr>
<td>haggis</td>
<td>£3.10</td>
</tr>
</tbody>
</table>

- **menu**: item -> price
- **customer operations**:
  - query: item -> menu -> price
    - is this different to menu...?
  - order: {item * quantity}+ -> menu -> price
Example: functionality

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>chips</td>
<td>£2.50</td>
</tr>
<tr>
<td>haddock</td>
<td>£6.50</td>
</tr>
<tr>
<td>cod</td>
<td>£5.45</td>
</tr>
<tr>
<td>pie</td>
<td>£2.75</td>
</tr>
<tr>
<td>fish cake</td>
<td>£1.20</td>
</tr>
<tr>
<td>chicken</td>
<td>£5.95</td>
</tr>
<tr>
<td>sausage</td>
<td>£1.25</td>
</tr>
<tr>
<td>black pudding</td>
<td>£2.90</td>
</tr>
<tr>
<td>haggis</td>
<td>£3.10</td>
</tr>
</tbody>
</table>

- customer: operations:
  - query: item -> menu -> price
    - is this different to menu...?
  - order: {item * quantity}+ -> menu -> price
Example: functionality

- chips  £2.50
- haddock  £6.50
- cod  £5.45
- pie  £2.75
- fish cake  £1.20
- chicken  £5.95
- sausage  £1.25
- black pudding  £2.90
- haggis  £3.10

- are these the same language or two languages?
- do we need to distinguish shop & customer mode?
  - yes – different gross functionalites
Example: functionality

• what happens if action fails?
• return success/fail message
  – add: \((\text{item} \to \text{price}) \to \text{menu} \to \text{menu} \ast \text{message}\)
  – delete: \(\text{item} \to \text{menu} \to \text{menu} \ast \text{message}\)
  – change: \((\text{item} \to \text{price}) \to \text{menu} \to \text{menu} \ast \text{message}\)
  – query: \(\text{item} \to \text{menu} \to \text{price} \ast \text{message}\)
  – order: \(\{\text{item} \ast \text{quantity}\}^{+} \to \text{menu} \to \text{price} \ast \text{message}\)
  • multiple messages for multiple items
Example: representations

- item & message == string
- message* == list of string
- price & quantity == integer
- menu & order == list of string & integer tuples

```
menu =
[("chips", 250),
 ("haddock", 650),
 ("cod", 545)]
...]
```
Example: functions

add: item * price -> menu -> menu * message

addM :: (String,Int) ->

    [(String,Int)] ->

    ([(String,Int)],String)

addM (item,price) menu =
    ((item,price):menu,"success")

• do we care if item in menu already...?
Example: functions

delete: item -> menu -> menu * message

deletem :: String -> [(String,Int)] -> 
       ([(String,Int)],String)
deletem item [] = ([],"can't find "++item)
deletem item ((item1,pricel):rest) = 
   if item==item1 
   then (rest,"success")
   else 
      let (menu,message) = deletem item rest 
      in ((item1,pricel):menu,message)
Example: functions

\[
\text{change: (item -> price) -> menu -> menu * message}
\]

\[
\text{changeM :: (String,Int) -> [(String,Int)] ->}
\]
\[
\text{([[(String,Int)],String])}
\]

\[
\text{changeM (item,\_ ) [] = ([],"can't find "++item)}
\]

\[
\text{changeM (item,price) ((item1,price1):rest) =}
\]
\[
\text{if item==item1}
\]
\[
\text{then ((item,price):rest,"success")}
\]
\[
\text{else}
\]
\[
\text{let (menu,message) =}
\]
\[
\text{changeM (item,price) rest}
\]
\[
\text{in ((item1,price1):menu,message)}
\]
Example: functions

query: item -> menu -> menu * message

queryM :: String -> [(String,Int)] -> (Int,String)
queryM item [] = (0,"can't find "++item)
queryM item ((item1,price1):rest) =
  if item==item1
  then (price1,"success")
  else queryM item rest
Example: functions

order: \{item \,*\, quantity\}\^+ \rightarrow menu \rightarrow price \,*\, message\^*

\text{buyM} :: (String,Int) \rightarrow [(String,Int)] \rightarrow (Int, String)

\text{buyM (item,quantity) menu =}
\text{let (price,mm) = queryM item menu}
\text{in (price*quantity,mm)}
Example: functions

**order**: \{item \* quantity\}^+ -> menu -> price \* message

\[
buysM :: [(String,Int)] \rightarrow [(String,Int)] \rightarrow (Int,[String])
\]

\[
buysM [] menu = (0,[])
\]

\[
buysM (h:t) menu =
    let (t1,mm1) = queryM h menu
    in
    let (t2,mm2) = buysM t menu
    in (t1+t2,mm1:mm2)
\]
Example: abstract syntax/data type

\[ s \rightarrow s \mid \text{add item price} \mid \text{delete item} \mid \text{change item price} \]

\[ \text{SAST} = \text{ADD (String,Int)} \mid \text{DELETE (String,Int)} \mid \text{CHANGE (String,Int)} \]

\[ ss = [\text{SAST}] \]
Example: abstract syntax/data type

c -> c c |
  query item |
  buy items

items -> item quantity | items items

CAST = QUERY String |
  BUY [(String,Int)]

c c == [CAST]
Example: semantics

\[ \text{mShop: } s \rightarrow \text{menu} \rightarrow \text{menu} \ast \text{message} \ast \]
\[ \text{mCustomer: } c \rightarrow \text{menu} \rightarrow \text{price} \ast \ast \text{message} \ast \]

- multiple prices from queries and orders

• details left as exercise ^\text{\textdagger}

SPLV 2019 University of Strathclyde
Example: interpreters

doShop :: SAST -> [(String,Int)] ->
    ([(String,Int)],String)
doShop SHOW menu = (menu,showM menu)
doShop (ADD(item,price)) menu =
    addM (item,price) menu
doShop (DELETE item) menu =
    deleteM item menu
doShop (CHANGE(item,price)) menu =
    changeM (item,price) menu
Example: interpreters

doSops :: [SAST] -> [(String,Int)] ->
        ([(String,Int)],[String])

   doShops [] menu = (menu,[])
   doShops (h:t) menu =
       let (m1,mm1) = doShop h menu
           in
           let (m2,mm2) = doShops t m1
               in (m2,mm1:mm2)
Example: interpreters

doCustomer :: CAST -> [(String,Int)] -> ([Int],[String])
  
doCustomer (QUERY item) menu =  
    let (p,mm) = queryM item menu  
    in ([p],[mm])

  
doCustomer (BUY l) menu =  
    let (p,mm) = buysM l menu  
    in ([p],mm)
Example: interpreters

doCustomers :: [CAST] -> [(String,Int)] -> ([Int],[String])
doCustomers [] menu = ([],[])
doCustomers (h:t) menu = 
  let (p1,mm1) = doCustomer h menu 
in
  let (p2,mm2) = doCustomers t menu 
in (p1++p2,mm1++mm2)
Example: concrete syntax

shops -> shop | shop ; shops
shop -> add identifier price |
       delete identifier |
       change identifier price

customers -> customer | customer ; customers
customer -> query identifier | buy items
items -> item | item + items
item -> identifier * quantity
Example: shop

menu:

[(“chips”, 250), (“cod”, 575), (“haddock”, 685)]

shop:

“add pie 325;
change chicken 795;
change cod 795;
delete chicken;
delete haddock”
Example: shop

AST:

[ADD pie 325,
  CHANGE chicken 795,
  CHANGE cod 795,
  DELETE chicken,
  DELETE haddock]

new menu: ["pie",325], ["chips",250],
  ["cod",795]

messages:

["success","can't find chicken", "success",
  "can't find chicken","success"]
Example: customer

menu:

[ ("pie", 325), ("chips", 250), ("cod", 795) ]

customer:

"query chicken;
query cod;
buy cod*3+chicken*3+chips*4"
Example: customer

AST:

```
[QUERY "chicken",
 QUERY "cod",
 BUY [("cod",3),("chicken",3),("chips",4)]
```

prices:

```
[0, 795, 3385]
```

messages:

```
["can't find chicken", "success", "success", "can't find chicken", "success"]
```