An Overview of Domain Specific Languages

Laurence Tratt
http://tratt.net/laurie/

Middlesex University

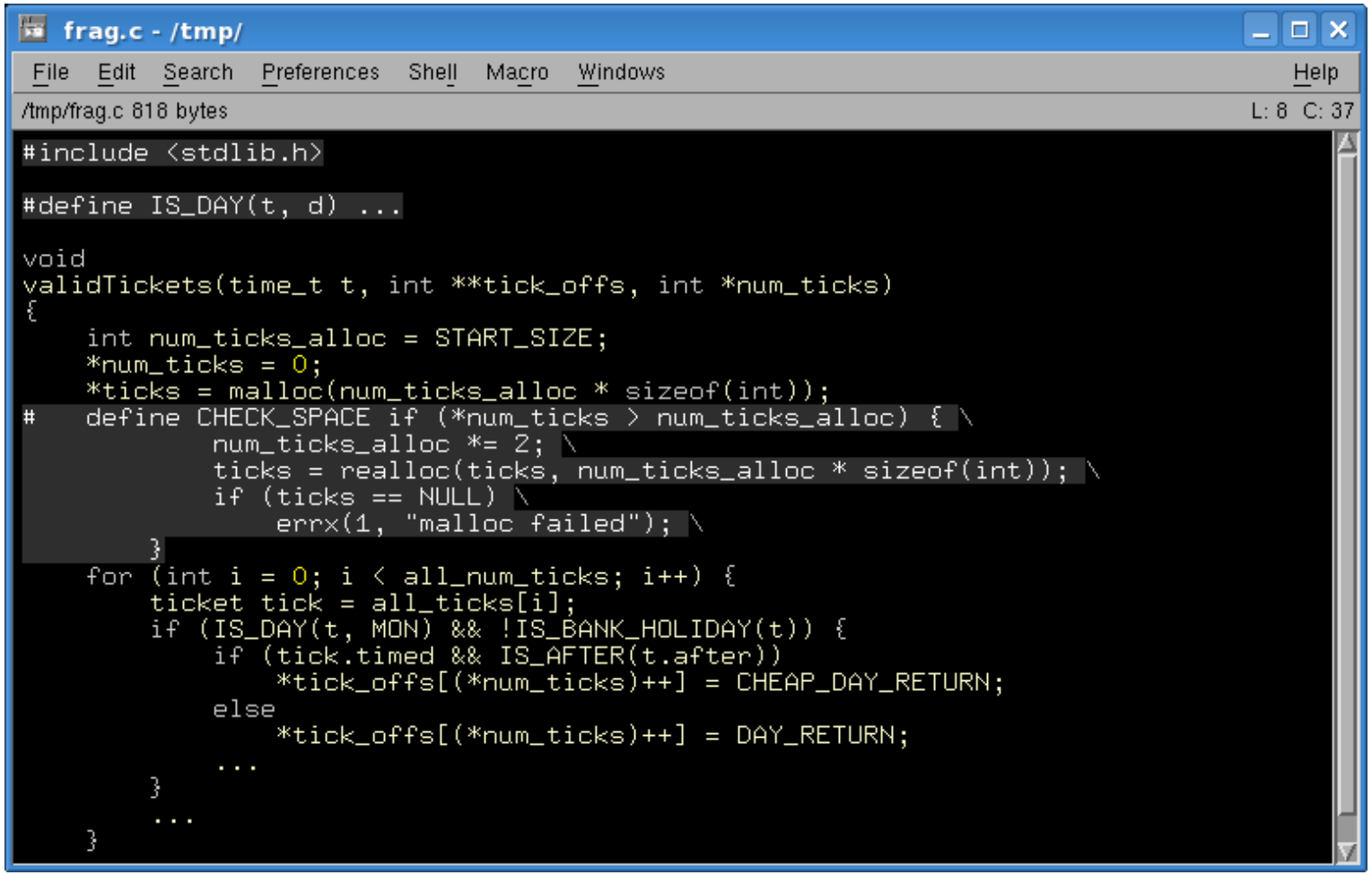
2010/06/09

A question

What's this?

A question

What's this?



A question

What's this?

frag.c - /tmp/

File Edit Search Preferences Shell Macro Windows Help

/tmp/frag.c 818 bytes L: 8 C: 37

```
#include <stdlib.h>

#define IS_DAY(t, d) ...

void
validTickets(time_t t, int **tick_offs, int *num_ticks)
{
    int num_ticks_alloc = START_SIZE;
    *num_ticks = 0;
    *ticks = malloc(num_ticks_alloc * sizeof(int));
#   define CHECK_SPACE if (*num_ticks > num_ticks_alloc) { \
            num_ticks_alloc *= 2; \
            ticks = realloc(ticks, num_ticks_alloc * sizeof(int)); \
            if (ticks == NULL) \
                errx(1, "malloc failed"); \
        }
    for (int i = 0; i < all_num_ticks; i++) {
        ticket tick = all_ticks[i];
        if (IS_DAY(t, MON) && !IS_BANK_HOLIDAY(t)) {
            if (tick.timed && IS_AFTER(t.after))
                *tick_offs[(*num_ticks)++] = CHEAP_DAY_RETURN;
            else
                *tick_offs[(*num_ticks)++] = DAY_RETURN;
            ...
        }
        ...
    }
```

Is it a language for computers or a language for railway timetables?

The situation

- To express a solution we need a language.

The situation

- To express a solution we need a language.
- On computers we turn to General Purpose Languages (GPLs)—e.g. Java, C#(), C++, Python, Ruby...

The situation

- To express a solution we need a language.
- On computers we turn to General Purpose Languages (GPLs)—e.g. Java, C#(), C++, Python, Ruby...
- For new or unusual problems, GPLs are nearly always great.
- But not always for repetitive tasks. Why?

Why do we have GPLs?

- Let's take Java.
- Main features: packages, classes, functions, static types, garbage collection, variables, `if`, `while`, `for`, and so on.

Why do we have GPLs?

- Let's take Java.
- Main features: packages, classes, functions, static types, garbage collection, variables, `if`, `while`, `for`, and so on.
- Really: building blocks.

Building blocks

- Virtually anything can be built with them...

Photo: David Iliff (licence)

Building blocks

- ...but it can be repetitive.

Photo: Mark Murphy (licence)

GPLs summary

- *Low level* building blocks.
- Virtually any task will need some (often all) of the building blocks.

GPLs summary

- *Low level* building blocks.
- Virtually any task will need some (often all) of the building blocks.
- But few naturally map onto them.

GPLs summary

- *Low level* building blocks.
- Virtually any task will need some (often all) of the building blocks.
- But few naturally map onto them.
- Very general; jacks of all trades, masters of none.
- The railway timetable uses only a tiny fraction of a GPLs power...

My GPL is better than yours

- But wait—my favourite language is better than Java!

My GPL is better than yours

- But wait—my favourite language is better than Java!

My GPL is better than yours

- But wait—my favourite language is better than Java!

(l-r) Java, C++, Python, C#, Haskell

Source: Library & Archives Canada (licence)

My GPL is better than yours

- But wait—my favourite language is better than Java!
- GPLs are nearly all extremely similar.
- We magnify small differences for cultural reasons.
- They're all jack of all trades, master of none.

DSLs—the basic idea

- DSL: a small language targetted at a specific class of problems.
- Allows you to specify repetitive tasks with small amounts of variation.
- ‘Do one thing and do it well.’

DSL examples

- SQL (databases)

Rev_Geo.py - /home/ltratt/work/elbatrop/src/locator/data/osm/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/work/elbatrop/src/locator/data/osm/Rev_Geo.py 23949 bytes L: 436 C: 0

```
            if way1_id == highway_id and \
              dst1 + _DISTANCE_FROM_OTHER_HIGHWAY / Geo.metres_at_latitude(lat) < dst2:
                found = True

        if found:
            related.append(["W", way_id, dst])

    # Node case

    c.execute("""
      SELECT
        nodes.id, ST_Distance(ST_GeomFromEWKT(%(item_ll)s), nodes.geom) AS dst,
        ST_AsEWKT(nodes.geom)
      FROM nodes, node_tags AS nt1, node_tags AS nt2
      WHERE
        ST_DWithin(ST_GeomFromEWKT(%(item_ll)s), nodes.geom, %(max_dst)s)
        AND nt1.node_id=nodes.id AND nt1.k='building'
        AND nt2.node_id=nodes.id AND (nt2.k='addr:housenumber' OR nt2.k='addr:housename')
      ORDER BY dst
      """, dict(highway_name=highway_name, item_ll=lls, \
      max_dst=_MAX_DISTANCE_OF_BUILDING / Geo.metres_at_latitude(lat)))
    c2 = db.cursor()
    for node_id, dst, node_geom in c.fetchall():
        c2.execute("""
          SELECT
            ways.id, ST_Distance(ST_GeomFromEWKT(%(node_geom)s), ways.linestring) AS dst
          FROM ways, way_tags
          WHERE
            ST_DWithin(ST_GeomFromEWKT(%(node_geom)s), ways.bbox, %(max_dst)s)
            AND way_tags.way_id=ways.id AND way_tags.k='highway'
          ORDER BY dst
          LIMIT 2
          """, dict(node_geom=node_geom, \
          max_dst=_MAX_DISTANCE_OF_BUILDING / Geo.metres_at_latitude(lat)))
        assert c2.rowcount > 0 and c2.rowcount <= 2
        found = False
        if c2.rowcount == 1:
            found = True
        else:
            way1_id, dst1 = c2.fetchone()
            way2_id, dst2 = c2.fetchone()
```

DSL examples

- `make` (software builds)

Makefile.Stdlib - /home/ltratt/src/converge/current/lib/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/src/converge/current/lib/Makefile.Stdlib 1753 bytes L: 46 C: 0

```
%.cvb: %.cv
    ${CONVERGE_VM} ${CONVERGEC} -I Stdlib -o $@ $<

%.cvb: %
    ${CONVERGE_VM} ${CONVERGEC} -o $@ $<


all: Stdlib.cvl

minimal: ${MINIMAL_OBJS}

install: all
    ${INSTALL} -d ${DESTDIR}${conlibdir}
    ${INSTALL} -c -m 444 Stdlib.cvl ${DESTDIR}${conlibdir}


ifdef TARGET
CROSS_OBJS = ${ALL_OBJS:.cvb=.${TARGET}.cvb}

%.${TARGET}.cvb: %.cv
    ${CONVERGE_VM} ${CONVERGEC} -T ${TARGET} -I ${CONVERGE_COMPILER_DIR} -o $@ $<

%.${TARGET}.cvb: %
    ${CONVERGE_VM} ${CONVERGEC} -T ${TARGET} -o $@ $<

cross: ${CROSS_OBJS}
    ${CONVERGE_VM} ${CONVERGEL} -l -T ${TARGET} -o Stdlib.${TARGET}.cvl Stdlib.${TARGET}.cvb $

cross-clean:
    rm -f ${CROSS_OBJS} Stdlib.${TARGET}.cvl
endif


Stdlib.cvl: ${ALL_OBJS}
    ${CONVERGE_VM} ${CONVERGEL} -l -o Stdlib.cvl Stdlib.cvb ${ALL_OBJS}

clean:
    rm -f ${ALL_OBJS} Stdlib.cvl
```

Hardware DSLs

- Question: are DSLs only for low-level software activities?

Hardware DSLs

- Question: are DSLs only for low-level software activities?
- Verilog: hardware description language.

```
module counter (clk,rst,enable,count);
input clk, rst, enable;
output [3:0] count;
reg [3:0] count;

always @ (posedge clk or posedge rst)
if (rst) begin
  count <= 0;
end else begin : COUNT
  while (enable) begin
    count <= count + 1;
    disable COUNT;
  end
end

endmodule
```

Source: Deepak Kumar Tala

Why would we want DSLs?

- DSLs are good when we do the same type of task repeatedly.
- But is that it?

Consideration 1: accessibility

- Programming is how we tell computers what to do.

Consideration 1: accessibility

- Programming is how we tell computers what to do.
- Many (most?) people struggle with programming...
- [c.f. the huge failure rates in undergrad programming.]

Consideration 1: accessibility

- DSLs can remove complex confusing features.

Consideration 1: accessibility

- DSLs can remove complex confusing features.
- ```
  income tax {
    2010-2011 {
      allowance {
        age < 65: £6,475
        age >= 65 and age <= 74: £9,490
        age > 74: £9,640

        reduction: if income > £100,000 then
          max(0, allowance - ((income - £100,000) / 2))
      }
    }
  }
  ```

Tax rules source: HMRC
  ```

Consideration 1: accessibility

- DSLs can remove complex confusing features.
- ```
  income tax {
    2010-2011 {
      allowance {
        age < 65: £6,475
        age >= 65 and age <= 74: £9,490
        age > 74: £9,640

        reduction: if income > £100,000 then
          max(0, allowance - ((income - £100,000) / 2))
      }
    }
  }
  ```

  Tax rules source: HMRC

Pros / cons:

+ Can allow non-programmers to do programming-like things.
  ```

Consideration 1: accessibility

- DSLs can remove complex confusing features.
- ```
  income tax {
    2010-2011 {
      allowance {
        age < 65: £6,475
        age >= 65 and age <= 74: £9,490
        age > 74: £9,640

        reduction: if income > £100,000 then
          max(0, allowance - ((income - £100,000) / 2))
      }
    }
  }
  ```

  Tax rules source: HMRC

Pros / cons:

+ Can allow non-programmers to do programming-like things.
- Sometimes complexity is fundamental.
  ```

Consideration 2: implementation flexibility

- Virtually all programming is done in imperative languages.

Consideration 2: implementation flexibility

- Virtually all programming is done in imperative languages.
- Advantage: explicitness.

Consideration 2: implementation flexibility

- Virtually all programming is done in imperative languages.
- Advantage: explicitness. Disadvantage: explicitness.

Consideration 2: implementation flexibility

- Virtually all programming is done in imperative languages.
- Advantage: explicitness. Disadvantage: explicitness.
- DSLs are an abstraction over a domain.

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

- How do you make parallelized versions of each?

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

- How do you make parallelized versions of each?
- C: rewrite your program (pthreads etc.).

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

- How do you make parallelized versions of each?
- C: rewrite your program (pthreads etc.).
- SQL: a cleverer SQL implementation.

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

- How do you make parallelized versions of each?
- C: rewrite your program (pthreads etc.).
- SQL: a cleverer SQL implementation.

Pros / cons:

+ Moves the burden from programmer to language implementer.

Consideration 2: implementation flexibility

- SQL:

```
SELECT * FROM nodes WHERE node.parent=NULL;
```

- C:

```
table *nodes = get_table(db, "nodes");
cursor *c = mk_cursor(nodes);
row *r;
results res = mk_results();
while ((r = get_next(c)) != null) {
  if (get_column(r, "parent") == null)
    add_result(res, r);
}
```

- How do you make parallelized versions of each?
- C: rewrite your program (pthreads etc.).
- SQL: a cleverer SQL implementation.

Pros / cons:

+ Moves the burden from programmer to language implementer.
- Over-abstraction can preclude some reasonable programs.

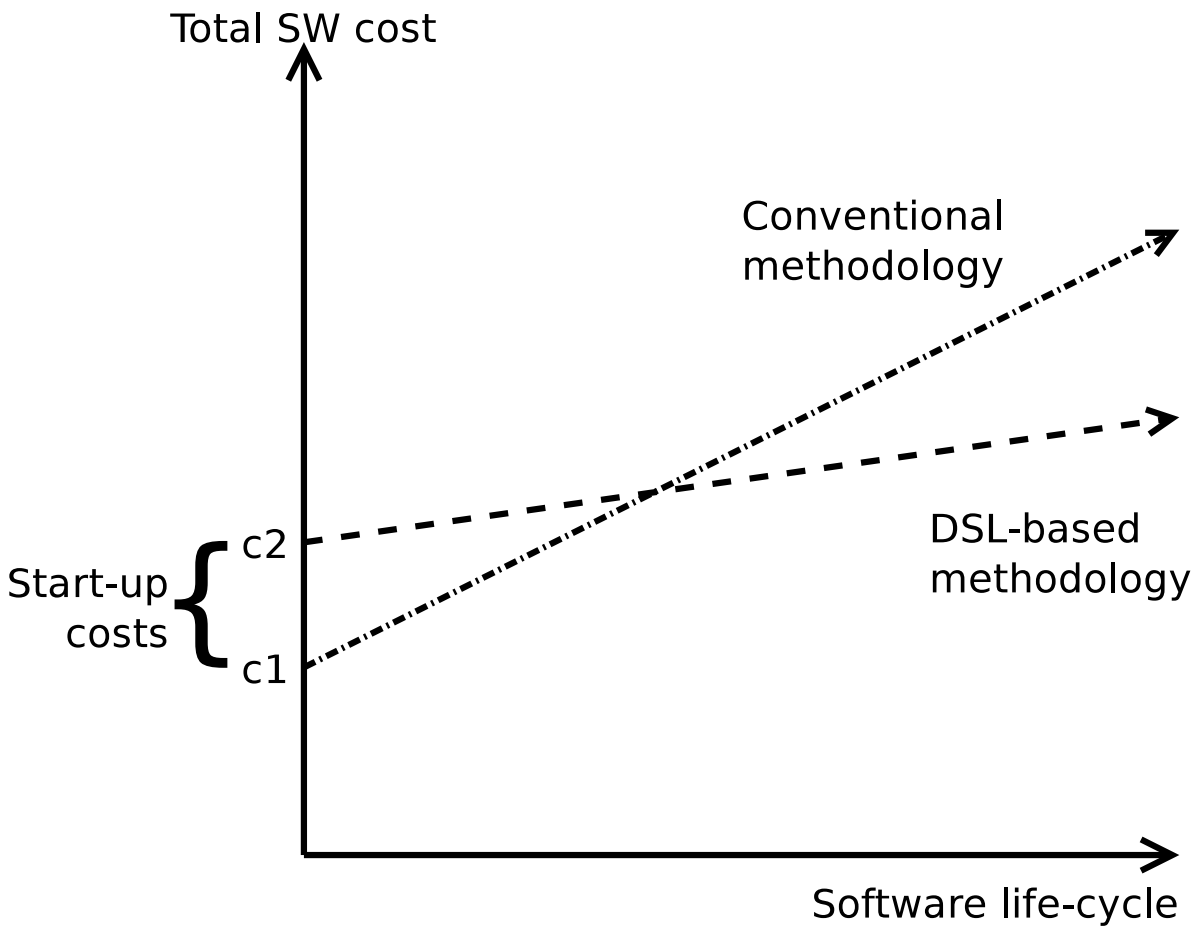
Consideration 3: Economics

- The bottom line: does it save money?

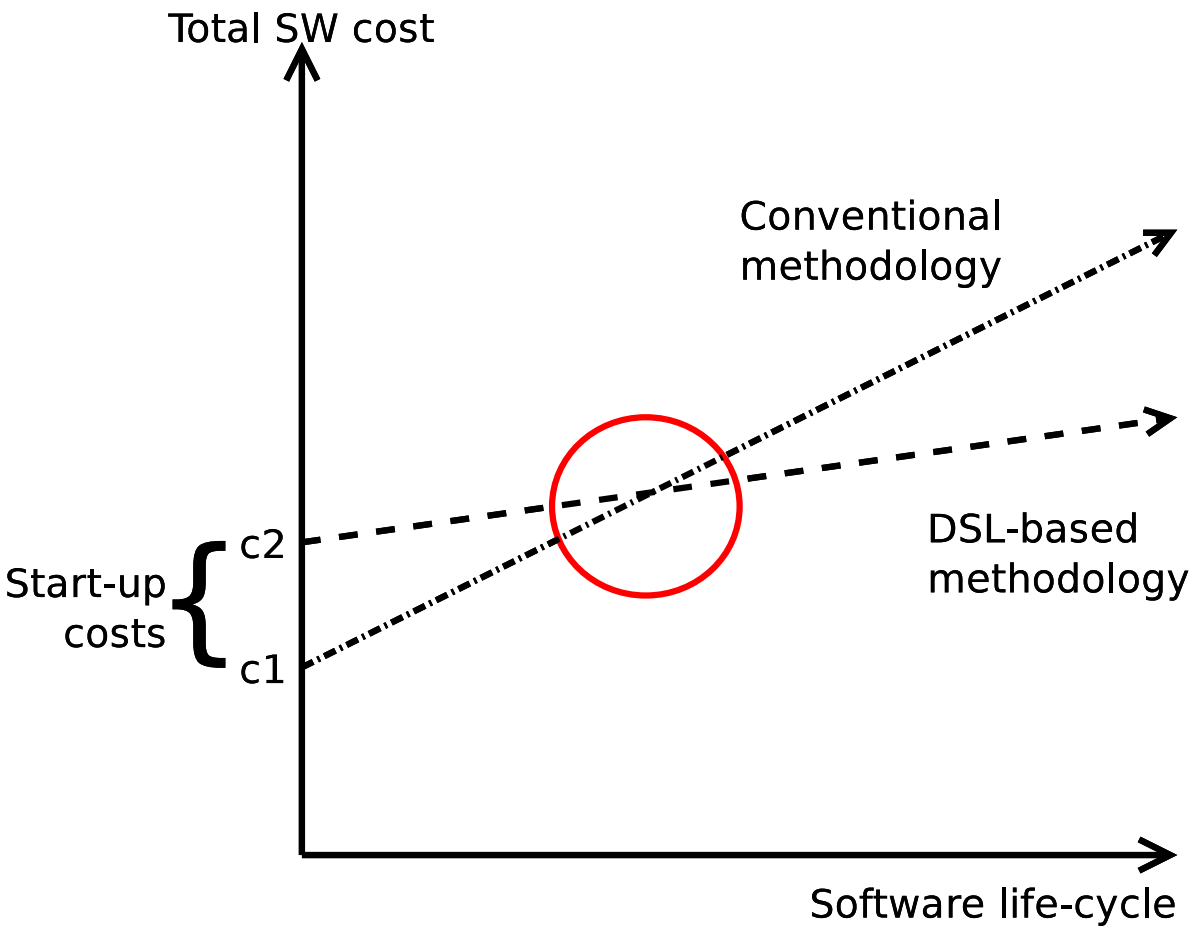
Consideration 3: Economics

- The bottom line: does it save money?
- If you're using someone elses DSL: almost certainly yes.
- But if you need to build a DSL: it depends.

Consideration 3: Economics

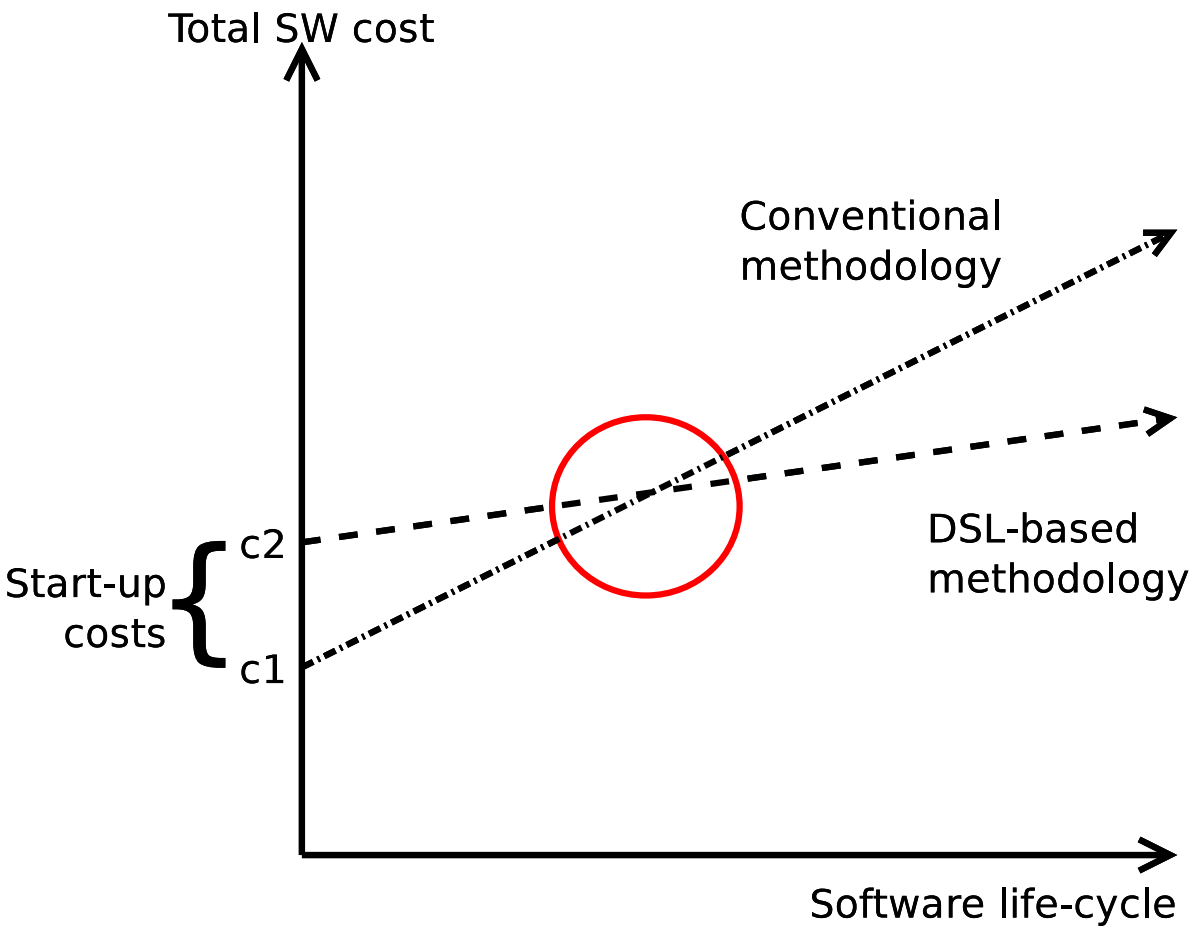


Consideration 3: Economics



Source: P. Hudak 'Modular domain specific languages and tools'

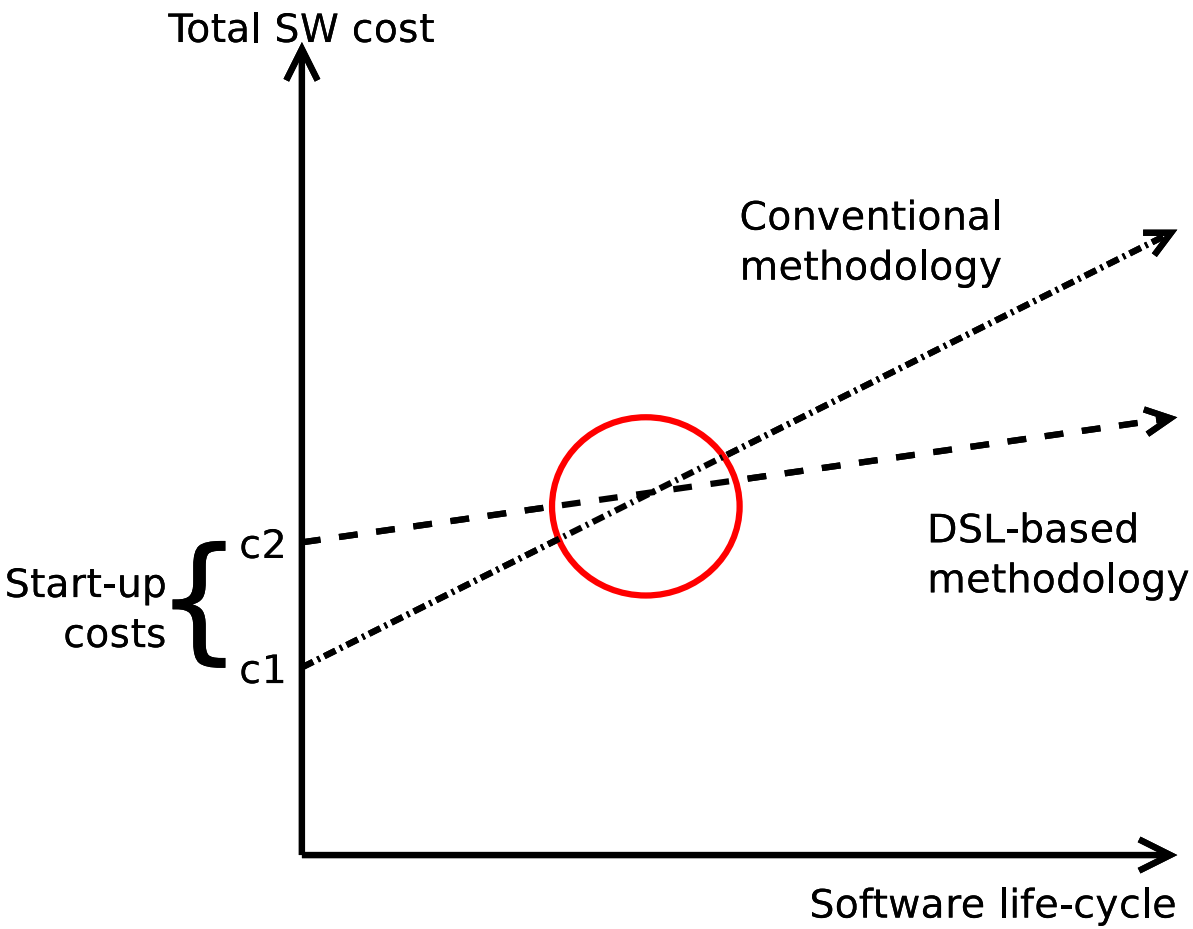
Consideration 3: Economics



Source: P. Hudak 'Modular domain specific languages and tools'

+ It can save *serious* amounts of money.

Consideration 3: Economics



Source: P. Hudak 'Modular domain specific languages and tools'

+ It can save *serious* amounts of money.
- Short-term hit for long-term gain.

What defines a DSL?

- [Inherently subjective and ill-defined. But...]

What defines a DSL?

- [Inherently subjective and ill-defined. But...]
- Has a well-defined problem domain.

What defines a DSL?

- [Inherently subjective and ill-defined. But...]
- Has a well-defined problem domain.
- Has its own syntax.
- [Practically speaking: its own implementation]

What DSLs aren't

- Haskell and Ruby people talk about 'internal DSLs'.
- Just a [clever?] way of using libraries.
- IMHO: not DSLs. Better called fluent interfaces.

DSL flavours

- `make`: standalone

Makefile.Stdlib - /home/ltratt/src/converge/current/lib/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/src/converge/current/lib/Makefile.Stdlib 1753 bytes L: 46 C: 0

```
%.cvb: %.cv
    ${CONVERGE_VM} ${CONVERGEC} -I Stdlib -o $@ $<

%.cvb: %
    ${CONVERGE_VM} ${CONVERGEC} -o $@ $<


all: Stdlib.cvl

minimal: ${MINIMAL_OBJS}

install: all
    ${INSTALL} -d ${DESTDIR}${conlibdir}
    ${INSTALL} -c -m 444 Stdlib.cvl ${DESTDIR}${conlibdir}


ifdef TARGET
CROSS_OBJS = ${ALL_OBJS:.cvb=.${TARGET}.cvb}

%.${TARGET}.cvb: %.cv
    ${CONVERGE_VM} ${CONVERGEC} -T ${TARGET} -I ${CONVERGE_COMPILER_DIR} -o $@ $<

%.${TARGET}.cvb: %
    ${CONVERGE_VM} ${CONVERGEC} -T ${TARGET} -o $@ $<
I
cross: ${CROSS_OBJS}
    ${CONVERGE_VM} ${CONVERGEL} -l -T ${TARGET} -o Stdlib.${TARGET}.cvl Stdlib.${TARGET}.cvb $

cross-clean:
    rm -f ${CROSS_OBJS} Stdlib.${TARGET}.cvl
endif


Stdlib.cvl: ${ALL_OBJS}
    ${CONVERGE_VM} ${CONVERGEL} -l -o Stdlib.cvl Stdlib.cvb ${ALL_OBJS}

clean:
    rm -f ${ALL_OBJS} Stdlib.cvl
```

DSL flavours

- SQL: embedded, syntactically distinct, run-time

Rev_Geo.py - /home/ltratt/work/elbatrop/src/locator/data/osm/

File Edit Search Preferences Shell Macro Windows Help

/home/ltratt/work/elbatrop/src/locator/data/osm/Rev_Geo.py 23949 bytes L: 436 C: 0

```
            if way1_id == highway_id and \
              dst1 + _DISTANCE_FROM_OTHER_HIGHWAY / Geo.metres_at_latitude(lat) < dst2:
                found = True

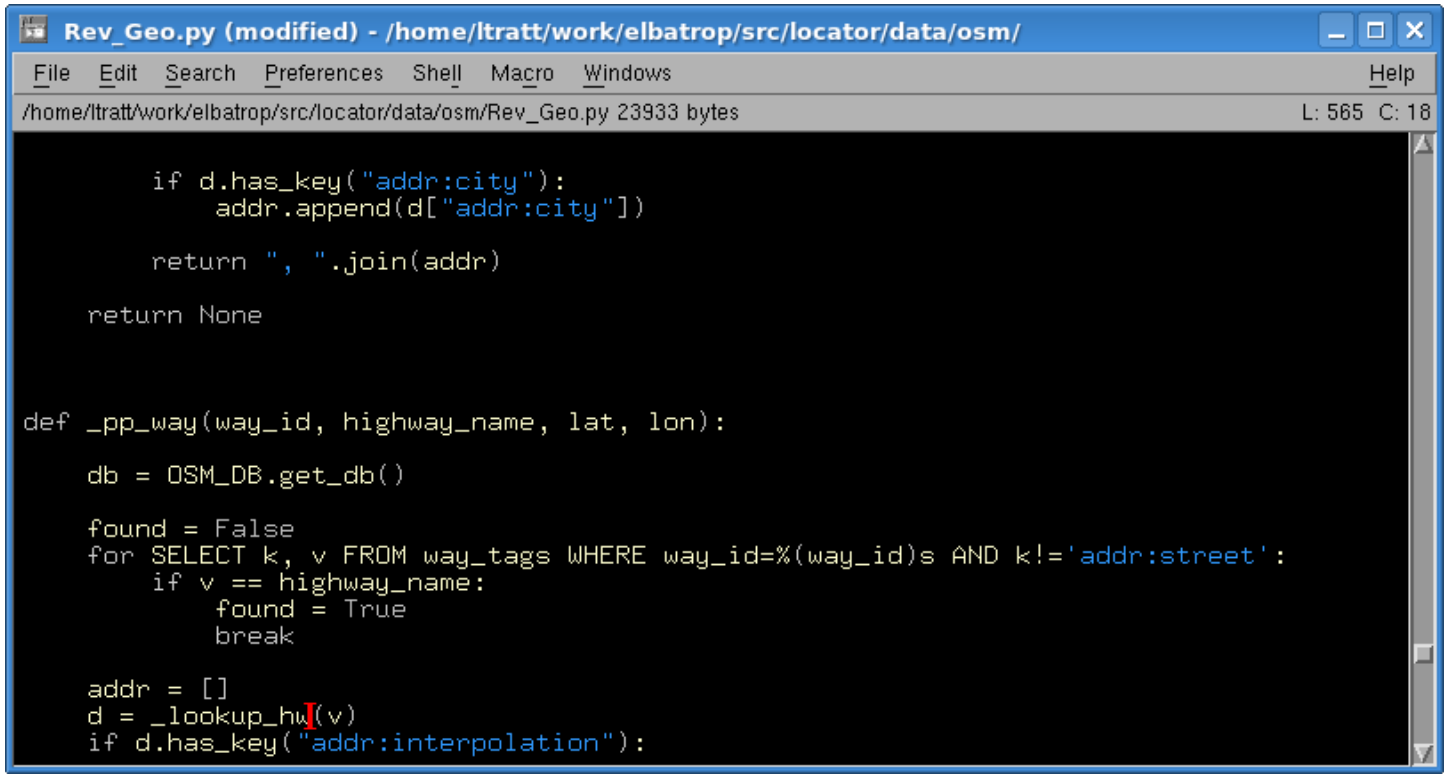
        if found:
            related.append(["W", way_id, dst])

    # Node case

    c.execute("""
      SELECT
        nodes.id, ST_Distance(ST_GeomFromEWKT(%(item_ll)s), nodes.geom) AS dst,
        ST_AsEWKT(nodes.geom)
      FROM nodes, node_tags AS nt1, node_tags AS nt2
      WHERE
        ST_DWithin(ST_GeomFromEWKT(%(item_ll)s), nodes.geom, %(max_dst)s)
        AND nt1.node_id=nodes.id AND nt1.k='building'
        AND nt2.node_id=nodes.id AND (nt2.k='addr:housenumber' OR nt2.k='addr:housename')
      ORDER BY dst
      """, dict(highway_name=highway_name, item_ll=lls, \
      max_dst=_MAX_DISTANCE_OF_BUILDING / Geo.metres_at_latitude(lat)))
    c2 = db.cursor()
    for node_id, dst, node_geom in c.fetchall():
        c2.execute("""
          SELECT
            ways.id, ST_Distance(ST_GeomFromEWKT(%(node_geom)s), ways.linestring) AS dst
          FROM ways, way_tags
          WHERE
            ST_DWithin(ST_GeomFromEWKT(%(node_geom)s), ways.bbox, %(max_dst)s)
            AND way_tags.way_id=ways.id AND way_tags.k='highway'
          ORDER BY dst
          LIMIT 2
          """, dict(node_geom=node_geom, \
          max_dst=_MAX_DISTANCE_OF_BUILDING / Geo.metres_at_latitude(lat)))
        assert c2.rowcount > 0 and c2.rowcount <= 2
        found = False
        if c2.rowcount == 1:
            found = True
        else:
            way1_id, dst1 = c2.fetchone()
            way2_id, dst2 = c2.fetchone()
```

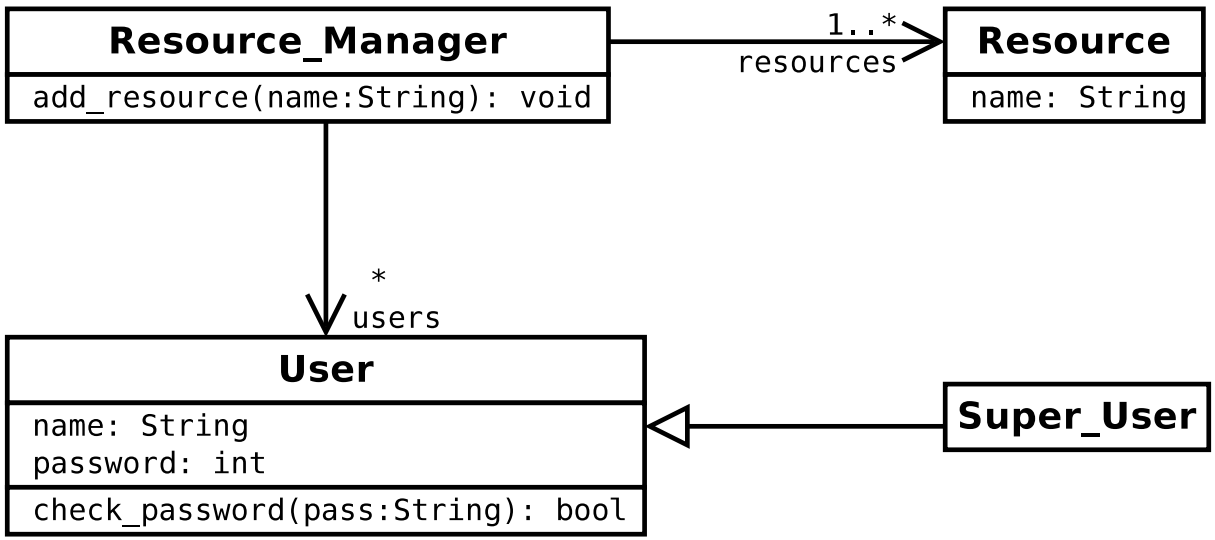
DSL flavours

- SQL: embedded, syntactically distinct, compile-time



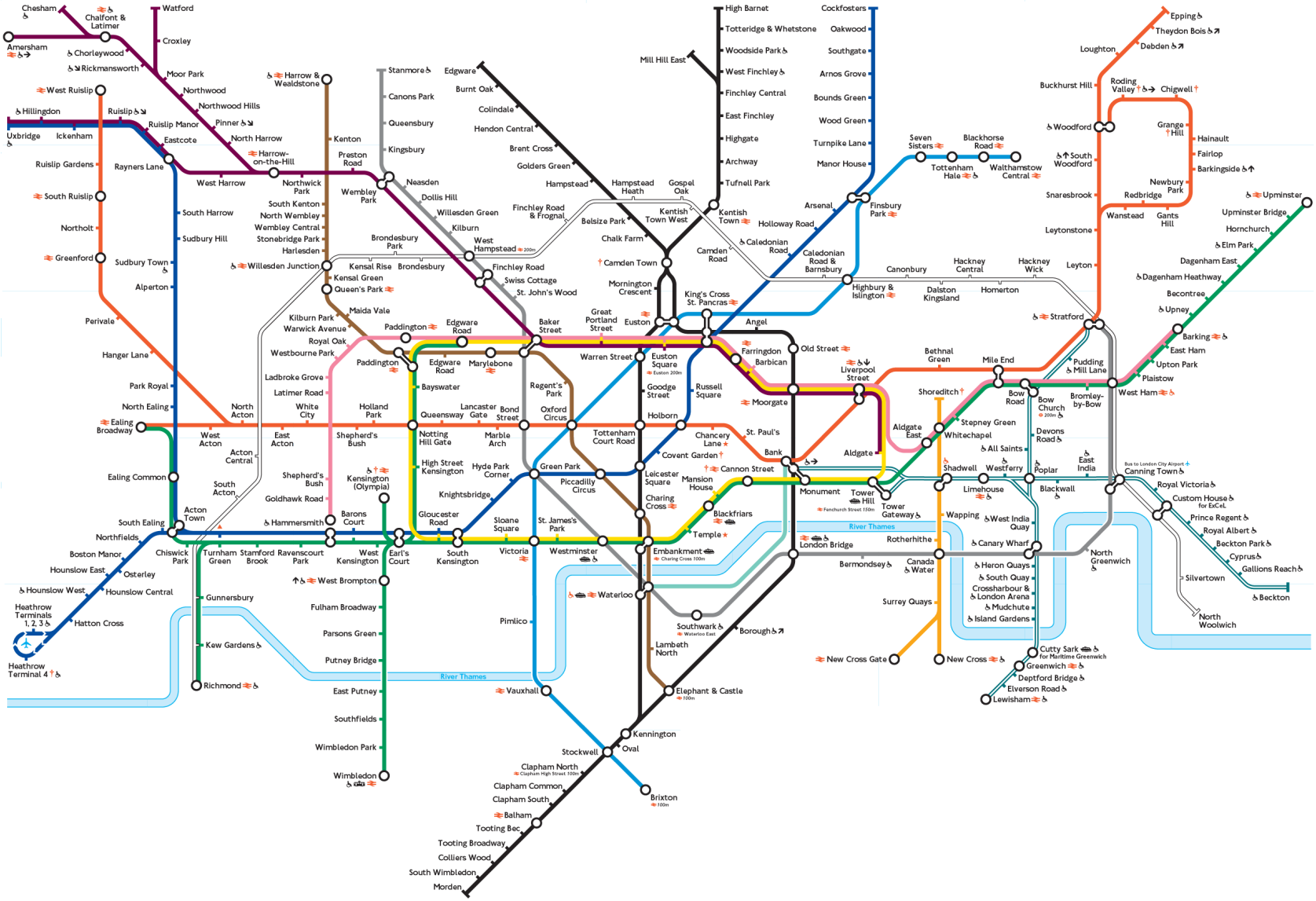
DSL flavours

- UML: diagrammatic



DSL flavours

- Metro systems: diagrammatic



How to build a DSL

- Assume you want to build a DSL.
- How? Who? How long?

Questions to ask when building a DSL

Questions to ask when building a DSL

- *Has someone else done it?*

Questions to ask when building a DSL

- *Has someone else done it?*
- *Who will use it?* Are there real users? Are they willing to use DSLs?

Questions to ask when building a DSL

- *Has someone else done it?*
- *Who will use it?* Are there real users? Are they willing to use DSLs?
- *How often will they use it?* Will it save money?

Questions to ask when building a DSL

- *Has someone else done it?*
- *Who will use it?* Are there real users? Are they willing to use DSLs?
- *How often will they use it?* Will it save money?
- *How will they use it?* Diagrammatic? Stand-alone? Embedded? ...

Questions to ask when building a DSL

- *Has someone else done it?*
- *Who will use it?* Are there real users? Are they willing to use DSLs?
- *How often will they use it?* Will it save money?
- *How will they use it?* Diagrammatic? Stand-alone? Embedded? ...
- *How will it integrate?* IDE plugin? Compiler extension? ...

Questions to ask when building a DSL

- *Has someone else done it?*
- *Who will use it?* Are there real users? Are they willing to use DSLs?
- *How often will they use it?* Will it save money?
- *How will they use it?* Diagrammatic? Stand-alone? Embedded? ...
- *How will it integrate?* IDE plugin? Compiler extension? ...
- *How will it evolve?* ...

DSL evolution

- The inevitable pattern: design a DSL for a small problem; users like it; want more; extend the DSL.
- Repeat ad nauseum.

DSL evolution

- The inevitable pattern: design a DSL for a small problem; users like it; want more; extend the DSL.
- Repeat ad nauseum.
- Result: DSLs tend to evolve messily.
- e.g. textual DSLs tend to end up resembling designed GPLs.
- This doesn't happen to GPLs: why not?

DSL evolution

- The inevitable pattern: design a DSL for a small problem; users like it; want more; extend the DSL.
- Repeat ad nauseum.
- Result: DSLs tend to evolve messily.
- e.g. textual DSLs tend to end up resembling designed GPLs.
- This doesn't happen to GPLs: why not?
- GPLs are so similar, we know how to do them.
- Each DSL tends to be a journey of exploration.

DSL implementation techniques

A representative sample:

- Stand alone.
- Converge (embedded, homogeneous).
- Stratego (embedded / standalone, heterogeneous).
- Intentional (embedded, heterogeneous).
- MPS (embedded, homogeneous).
- Xtext (standalone, heterogeneous).

Further reading

- Fowler: Language workbenches
- Stahl, Völter: Model-Driven Software Development
- Vasudevan, Tratt: Comparative study of DSL tools

Summary

- There are more DSLs in existence than we first think...
- ...and there will be a lot more.

Summary

- There are more DSLs in existence than we first think...
- ...and there will be a lot more.
- When DSLs are the right tool, they can lead to real savings.
- If you ask yourself the right questions, DSLs can work for you.