PADS: A Domain-Specific Language for Processing Ad Hoc Data

talk based on paper by
Kathleen Fisher¹ Robert Gruber²

¹AT&T Labs Research ²Google

November 10, 2005
Outline

1. Why worry about ad hoc data?

2. Current options for processing ad hoc data

3. What PADS does for you
Outline

1. Why worry about ad hoc data?
2. Current options for processing ad hoc data
3. What PADS does for you
Data and formatting

Massive amounts of data is generated and collected every day.

Formatted data is easier to understand and process.

Formats can either be standardized, or not standardized.
Standard vs nonstandard formats

Boundary not so clear, but...

there is certainly a difference in the size of the expected producer/user-base. Standard formats tend to have many applications that generate and process them. Standardized formats should be designed to expect a growing user-base.

Examples of data in standard formats:

- webpages in HTML
- pictures in JPEG
- XML
- data in databases
- The billions of lines of code you write every day?
Ad hoc data (data in nonstandard formats)

Claim/(fact?): Most data is actually stored in nonstandard formats. Here are some examples:

- AT&T accumulates 250-300GB/day of billing data
- Netflow data arrives at Cisco routers at over a GB/sec.
Common Log Format (CLF) for web servers

One record per line, with 7 fields:

```
HTTP/1.0" 200 941
```

The ’-’ character indicates missing data for that field
### Investment data

Date: 3/21/2005 1:00PM PACIFIC  
Investor’s Business Daily®  
Stock List Name: DAVE  

<table>
<thead>
<tr>
<th>Stock Symbol</th>
<th>Company Name</th>
<th>Price</th>
<th>Price Change</th>
<th>% Change</th>
<th>Volume</th>
<th>EPS</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AET</td>
<td>Aetna Inc</td>
<td>73.68</td>
<td>-0.22</td>
<td>0%</td>
<td>31%</td>
<td>64</td>
<td>93</td>
</tr>
<tr>
<td>GE</td>
<td>General Electric Co</td>
<td>36.01</td>
<td>0.13</td>
<td>0%</td>
<td>-8%</td>
<td>59</td>
<td>56</td>
</tr>
<tr>
<td>HD</td>
<td>Home Depot Inc</td>
<td>37.99</td>
<td>-0.89</td>
<td>-2%</td>
<td>63%</td>
<td>84</td>
<td>38</td>
</tr>
<tr>
<td>IBM</td>
<td>Intl Business Machines</td>
<td>89.51</td>
<td>0.23</td>
<td>0%</td>
<td>-13%</td>
<td>66</td>
<td>35</td>
</tr>
<tr>
<td>INTC</td>
<td>Intel Corp</td>
<td>23.50</td>
<td>0.09</td>
<td>0%</td>
<td>-47%</td>
<td>39</td>
<td>33</td>
</tr>
</tbody>
</table>

Data provided by William O’Neil + Co., Inc. © 2005. All Rights Reserved.  
Investor's Business Daily is a registered trademark of Investor's Business Daily, Inc.  
Reproduction or redistribution other than for personal use is prohibited.  
All prices are delayed at least 20 minutes.

### Records within another kind of record
Ad hoc data in chemistry

Context-free?

O=C([C@@H]2O[C@@]3(C)[C@](C)[C@](CO4)
(OC(C)=O)[C@H]4C[C@@H]3O)[H)[C@H]
(OC(C7=CC=CC=C7)=O)[C@@]1(O)[C@@](C)(C)C2=C(C)
[C@@H](OC([C@H](O)[C@@H](NC(C6=CC=CC=C6)=O)
C5=CC=CC=C5)=O)C1
DNS data

Binary data

00000000: 9192 d8fb 8480 0001 05d8 0000 0000 0872
00000010: 6573 6561 7263 6803 6174 7403 636f 6d00
00000020: 00fc 0001 c00c 0006 0001 0000 0e10 0027
00000030: 036e 7331 c00c 0a68 6f73 746d 6173 7465
00000040: 72c0 0c77 64e5 4900 000e 1000 0003 8400
00000050: 36ee 8000 000e 10c0 0c00 0f00 0100 000e
00000060: 1000 0a00 0a05 6c69 6e75 78c0 0c00 0c00
00000070: 0f00 0100 000e 1000 0c00 0a07 6d61 696c
00000080: 6d61 6ec0 0c00 0c00 0100 0100 000e 1000
00000090: 0487 cf1a 16c0 0c00 0200 0100 000e 1000
000000a0: 0603 6e73 30c0 0c00 0c00 0200 0100 000e
000000b0: 1000 02c0 2e03 5f67 63c0 0c00 2100 0100
000000c0: 0002 5800 1d00 0000 640c c404 7068 7973
000000d0: 0872 6573 6561 7263 6803 6174 7403 636f
..............r
research.att.com.
.............'
.ns1...hostmaster
r..wd.I........
6............
......linux....
..........mail
man............
.............
.ns0..........
......_gc...!
..X.....d...phys
.research.att.co
Small audience, big expectations

Groups that work with ad hoc data, no matter how small, still need tools that

• parse the data to load into applications
• visualize the data
• allow queries over the data
• convert the data (maybe to load into a database)
• detect errors in the data
• correct errors
• filter data
• combine data from multiple sources
Outline

1. Why worry about ad hoc data?

2. Current options for processing ad hoc data

3. What PADS does for you
Perl/AWK/Shell scripts/C

Pros

• flexible

Cons

• time investment
  • hand-code parser
  • hand-code error handling (if you even bother)
  • hand-code other tools (e.g., converters, viewers)
• error-prone
• difficult to maintain in the face of format changes
Lex + Yacc

Pros

- ?

Cons

- specify lexer and parser separately
- error handling inflexible
- still need to hand-code other tools (e.g., converters, viewers)

People don’t use Lex + Yacc for ad hoc data.
Outline

1. Why worry about ad hoc data?

2. Current options for processing ad hoc data

3. What PADS does for you
What we want

- Simple data description language
- Parser
  - should handle various encodings (ASCII, binary, etc.)
  - should not halt on errors, and
  - should track errors (syntactic + semantic)
  - generate useful in-memory representation
- Other tools
  - converter to other formats (e.g., XML, CSV)
  - statistical profiler
  - query interface
  - more?
- High performance
What we get from PADS

• Simple data description language
• Parser
  • handles ASCII, EBCDIC, and binary
  • does not halt on errors, and
  • tracks errors (syntactic + semantic) in a parse-descriptor struct
  • generates C structs, unions, etc., for in-memory rep.
• Other tools
  • convert/print to other formats (XML + XML Schema, CSV)
  • statistical profilers via accumulator programs
  • query interface
  • more in the future
• Faster than Perl programs
• Can stream data, rather than read it all into memory at once
PADS architecture

(Not to scale)
PADS architecture: a closer look

Note: Use masks to select only the features that are relevant to the application ⇒ increased performance
Data Description Language

**Primitives** (Pint32, Pdate, Pstring, Pip).

**Complex data** (Pstruct, Punion, Parray, Popt, Ptypedefs, Penum).

**Parameterized types** (e.g., Puint16_FW(:3:)).

Can attach predicates for error checking (see following example).
CLF: Checking HTTP version conformance

```c
Penum method_t {
    GET, PUT, POST, HEAD,
    DELETE, LINK, UNLINK
};

Pstruct request_t {
    '\"'; method_t meth;
    '\>'; Pstring(':') req_uri;
    '\>'; version_t version :
    chkVersion(version, meth);
    '\"';
};

int chkVersion(version_t v, method_t m) {
    if (((v.major == 1) && (v.minor == 1)) return 1;
    if (((m == LINK) || (m == UNLINK))
    return 0;
    return 1;
};
```
Siruis Data: Check that timestamps increase monotonically

```c
Pstruct event_t {
    Pstring(:'|':) state;
    '|' ;
    Puint32 tstamp;
};

Parray eventSeq {
    event_t[] : Psep('|') && Pterm(Pend);
} Pwhere {
    Pforall(i Pin [0..length-2] :
        {elts[i].tstamp <= elts[i+1].tstamp});
};
```
Conclusion

• They evaluate the work by showing how to construct parsers for a few formats and comparing the performance of generated tools to hand-coded Perl programs. PADS generated tools are no slower (actually faster in all 6 runs and 2x faster in 3 of the runs).

• Rather than standardize a format (e.g., HTML), then create tools, PADS can automatically generate tools as the format evolves. The only cost is the time to update the PADS description.

If you want to try it, go to http://www.padsproj.org
Discussion

• What other methods of evaluation would you like to see for work of this kind?
• The PADS group plans to add more auxiliary tool generators. Is it really useful, or do you imagine people building custom tools using only the generated parser?
• If people build custom tools, does that also build up data-format inertia?
• Do you feel that the PADS compiler has more opportunities to optimize the generated parser and tools than compiled hand-coded tools?