Automatic Tool Generation

for

Ad Hoc Data Processing

David Walker (PI)

Kathleen Fisher

Yitzhak Mandelbaum, Vivek Pai (Co-PI)

Princeton University & AT&T Research
Data, Data Everywhere

- **Software systems of all kinds generate vast amounts of data** -- often in *ad hoc* formats.

- **Ad hoc data is mostly structured:**
  - Not free text.
  - But not held in standard relational database.
  - Not necessary standardized (ie, not XML, CSV)
    - More non-standard legacy data formats than anything else?
<table>
<thead>
<tr>
<th>IP Address</th>
<th>Referrer</th>
<th>Timestamp</th>
<th>Request Method</th>
<th>Endpoint</th>
<th>Status Code</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>Value</td>
<td>Label</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------</td>
<td>------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9192</td>
<td>d8fb 8480 0001 05d8 0000 0000 0872</td>
<td>.................r</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6573</td>
<td>6561 7263 6803 6174 7403 636f 6d00</td>
<td>esearch.att.com.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00fc</td>
<td>0001 c00c 0006 0001 0000 0e10 0027</td>
<td>.................'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>036e</td>
<td>7331 c00c 0a68 6f73 746d 6173 7465</td>
<td>.ns1...hostmaste</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72c0</td>
<td>0c77 64e5 4900 000e 1000 0003 8400</td>
<td>r..wd.l.........</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36ee</td>
<td>8000 000e 10c0 0c00 0f00 0100 000e</td>
<td>6.................</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0a00 0a05 6c69 6e75 78c0 0c00 0c00</td>
<td>......linux.....</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0f00</td>
<td>0100 000e 1000 0c00 0a07 6d61 696c</td>
<td>............mail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6d61</td>
<td>6ec0 0cc0 0c00 0100 0100 000e 1000</td>
<td>man.................</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0487</td>
<td>c1fa 16c0 0c00 0200 0100 000e 1000</td>
<td>...................</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0603</td>
<td>6e73 30c0 0cc0 0c00 0200 0100 000e</td>
<td>..ns0.............</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>02c0 2e03 5f67 63c0 0c00 2100 0100</td>
<td>......_gc...!......</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>5800 1d00 0000 640c c404 7068 7973</td>
<td>.x.....d...phys</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0872</td>
<td>6573 6561 7263 6803 6174 7403 636f</td>
<td>.research.att.co</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[Term]
id: GO:0000001
name: mitochondrion inheritance
namespace: biological_process
def: "The distribution of mitochondria, including the mitochondrial genome, into daughter cells after mitosis or meiosis, mediated by interactions between mitochondria and the cytoskeleton." [PMID:10873824, PMID:11389764, SGD:mcc]
is_a: GO:0048308 ! organelle inheritance
is_a: GO:0048311 ! mitochondrion distribution

www.geneontology.org
Ad Hoc Data in Finance

HA00000000 START OF TEST CYCLE
aA00000001BXYZ U1AB000000400000100B00000004200
HL00000002 START OF OPEN INTEREST
d 00000003FZYX G1AB000000300000300000
HM00000004 END OF OPEN INTEREST
HE00000005 START OF SUMMARY
f 00000006NYZX
B1QB00052000120000070000B0000500000000520000
 00490000005100+00000100B0000000530000000525000000535000
HF00000007 END OF SUMMARY

www.opradata.com
Properties of Ad hoc Data

- Data arrives “as is” -- you don’t choose the format
- Documentation is often out-of-date or nonexistent.
- Data is buggy.
  - Missing data, “extra” data, …
  - Human error, malfunctioning machines, software bugs …
  - Errors are sometimes the most interesting portion of the data.
- Data sources often have high volume.
- Data can be created by malicious sources attempting to exploit software vulnerabilities
  - c.f. Ethereal network monitoring system
Project Goals

- **Automated generation of tools**
  - for continuous, real-time monitoring and historical access to widely disparate kinds of data

- **Widely applicable**
  - Operate over data in any legacy format

- **Simple adoption and ease-of-use**
  - Minimize programming effort

- **Robust in the face of errors**
  - And provide capabilities/programming model for diagnosing and processing errors
Approach: A Domain-Specific Programming Language

- **PADS/ML Language & Compiler System**
  - Declarative “data description language”
  - Supports ASCII, binary & mixed ad hoc data formats
  - Automatically generates a variety of programming libraries & stand-alone tools
  - Formal denotational semantics for reasoning about specification properties
PADS Compiler System

1. Input data description
2. PADS compiler
3. Generated libraries
4. Core PADS library
   - Profiling template
   - XML template
   - Format template
   - Custom user code
5. Data statistics
6. Formatted
   - XML
   - User specified

Code written per:
- Data source
- Tool
- Data source, application
Pads Parsing

- **data description** (type T)
- **parser**
- **compiler**
- **data rep** (type ~ T)
- **parse descriptor** (type ~ T)
- **user code**

Binary code: 01001001 00111
PADS Language

Data Descriptions look (largely) like a series of type definitions in your favourite programming language

- **Rich and extensible set of base types:**
  - Pint8, Pint32, ...
  - Pstring(‘|’), PstringFW(x)
  - Pdate, Ptime, Pip, ...

- **Dependent type constructors:**
  - (Dependent) records
  - (Dependent) ML-style datatypes
    - (ie: data-dependent recursive unions of records)
  - Lists
  - Parameterization: By description & value
  - Arbitrary (Data Dependent) Constraints
An Example Type Definition

```plaintext
ptype weblog = {
  host client;                      /- Client requesting service
  ''; auth_id remoteID;            /- Remote identity
  ''; auth_id auth;                /- Name of authenticated user
  " ["; Pdate(':') date;           /- Timestamp of request
  "] "]"; http_request request;   /- Request
  "; Puint16_FW(:3:) response;     /- 3-digit response code
  "; Puint32 contentLength;        /- Bytes in response
};
```

```
207.136.97.50 - - [15/Oct/1997] "GET /turkey.gif HTTP/1.0" 200 3013
```
PADS/ML Regulus Format:

```
ptype Timestamp = 
    Ptimestamp_explicit_FW(8, "%H:%M:%S", gmt)

ptype Pip = 
    Puint8 * '.' * Puint8 * '.' * Puint8 * '.' * Puint8

ptype (Alpha) Pnvp(p : string -> bool) = 
    { name : [name : Pstring('=') | p name]; '='; 
      value : Alpha }

ptype (Alpha) Nvp(name:string) = 
    Alpha Pnvp(fun s -> s = name)

ptype SVString = Pstring_SE(";/\|/\")

ptype Nvp_a = SVString Pnvp(fun _ -> true)

ptype Details = 
    { source : Pip Nvp("src_addr"); 
      dest : Pip Nvp("dest_addr"); 
      start_time : Timestamp Nvp("start_time"); 
      end_time : Timestamp Nvp("end_time"); 
      cycle_time: Puint32 Nvp("cycle_time") }

let checkCorr ra = ...

ptype Alarm = [x:Raw_alarm | checkCorr x]

ten Source = (Alarm,Peor,Peof) Plist
```

Sample Regulus Data:

```
2:3004092508||5001|dns1=abc.com;dns2=xyz.com|c=slow link;w=lost packets|INTERNATIONAL
3:|3004097201|5074|dns1=bob.com;dns2=alice.com|src_addr=192.168.0.10; 
dst_addr=192.168.23.10;start_time=1234567890;end_time=1234568000;cycle_time=17412|SPECIAL
```
Advantages Over “Ad Hoc” Methods

- **Big bang for user buck:**
  - 1 description ==> many tools
  - 1 format-independent tool ==> many data sources

- **Descriptions document data sources**
  - the documentation IS the tool generator

- **Descriptions are easy to write, easy to understand.**

- **Tools are robust**
  - Error handling code generated automatically; doesn’t clutter documentation.

- **Descriptions have formal semantics**
Up Next

- New sorts of specifications for data location, access mode & temporal availability
  - Rearchitect Vivek Pai’s CoMon infrastructure to generate monitoring and visualization tools from specs
- Learn descriptions from example data automatically
Progress Summary
[NSF support since summer 06]

- **PADS** [http://www.padsproj.org]
  - Defined tool generation architecture & formal semantics for PADS/ML [POPL 07, TFP 07]
  - First release of PADS/ML: Jan 07

- **CoMon** [http://comon.cs.princeton.edu]
  - New visualization tools & backend optimizations
  - Modularization of software infrastructure in preparation for using PADS-generated software components
  - Usage: 700+ ips; 30+ users accessing monitoring and visualization tools at any time
End
Existing Approaches

- **Lex/Yacc**
  - Over & Underkill.

- **Perl/C**
  - Code brittle with respect to changes in input format.
  - Analysis often ends up interwoven with parsing.
  - Error code, if written, swamps main-line computation. If not written, errors can corrupt “good” data.
  - Everything has to be coded by hand.

- **Packet description languages**
  - eg: PacketTypes, Datascript
  - Binary data
  - Focus on correct data.
CoMon Monitoring Framework

- **End Goal:** To generate custom monitoring components from high-level specifications
  - Another whole domain-specific collection of tools

- **Current CoMon Status**
  - **Data Volume**
    - 700 nodes, 300 experiments, 10 data points per experiment instance
  - **Interactivity**
    - 5 minute refresh interval, History stored over two days
    - Continuous & on-demand visualization mechanisms
  - **Usage**
    - 700 IPs; 30 users accessing visualization tools at any time
Approach & Technology

- **PADS/ML Language & Compiler System**
  - Declarative “data description language”
  - Supports ASCII, binary & mixed ad hoc data formats
  - Automatically generates a variety of programming libraries & stand-alone tools

- **CoMon systems monitoring infrastructure**
  - Current data volume: 700 nodes, 300 experiments, 10 data points/experiment
  - Interactivity: 5 minute refresh interval; history stored over two days
  - Infrastructure: data acquisition tools; simple query infrastructure; anomaly detection; multiple visualization modes
## CoMon Node-Centric

| Name | Address | SSH Status | Last CoTop | Kerna Ver | Boot State | Drift | CPU Speed | Last Min Load | Mem Siz Mem Act Mem Free Mem Avg Swap In Swap Out Disk In Disk Out Disk Usage GB Free Swap Used | FD Test | HW Limits | Tx Rate Rx Rate |
|------|---------|------------|------------|-----------|------------|-------|-----------|--------------|-------------|-------------------------------------------------|---------|-----------|---------------|
CoMon TreeMap Support

<table>
<thead>
<tr>
<th></th>
<th>nyud</th>
<th>princeton codeen</th>
<th>hplabs oasis</th>
<th>ucr slice4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>290.3 Mbps</td>
<td>244.0</td>
<td>80.8 -1.0</td>
<td>27.3 -1.2</td>
</tr>
<tr>
<td></td>
<td>2.0 dev</td>
<td>2.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>princeton coblitz</td>
<td>utah elab</td>
<td>wuerzburg redarlf</td>
<td></td>
</tr>
<tr>
<td></td>
<td>286.3</td>
<td>24927</td>
<td>7.3 23.9</td>
<td>1.5 13</td>
</tr>
<tr>
<td></td>
<td>-0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>149.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Note: The image contains a complex TreeMap with various data points and colors, indicating network usage or performance metrics across different systems.)
Progress Summary [Since Summer 06]

- **PADS** [www.padsproj.org]
  - Defined tool generation architecture & formal semantics [POPL 07, TFP 07]
  - First release of PADS/ML: Jan 07
  - Up Next: Support for distributed data sources
    - New abstractions for remote locations, access modes and schedule of data availability
  - Up Next: Automated inference of data descriptions

- **CoMon**
  - New visualization support & backend optimizations
  - Modularization of software infrastructure
  - Up Next: Automatic generation of data gathering
PADS: One Description, Many Tools

Data Description (Type T)

- Compiler
- XML translator
- Query engine
- Parser
- Printer
- Statistical analysis

Legend:
- Programming library
- Complete application
CoMon Slice Usage Totals (sort key: 15-min Transmit)

Part of the CoMom project
Updated Thu Mar 22 12:20:02 2007
Summary: By Node (long, short) By Slice (max, average, total, site) By Ports (all) By Site (all)
Problems (nodes, slices) View Auto Resources (CPU, Mem, BW) Efficiency (CPU, Mem) Usage (Slices, Slices, Nodes)

<table>
<thead>
<tr>
<th>#</th>
<th>Slice Name</th>
<th>1-min Transmit</th>
<th>15-min Transmit</th>
<th>1-min Receive</th>
<th>15-min Receive</th>
<th>Num Proc</th>
<th>Phys Mem MB</th>
<th>Virt Mem MB</th>
<th>CPU %</th>
<th>MEM %</th>
<th>Long Ports</th>
<th>Snap Ports</th>
<th># Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>princeton_cadre</td>
<td>225417</td>
<td>227919</td>
<td>66365</td>
<td>64718</td>
<td>1305</td>
<td>5570.1</td>
<td>8356.3</td>
<td>4957.0</td>
<td>463.8</td>
<td>2263</td>
<td>7064</td>
<td>268</td>
</tr>
<tr>
<td>2</td>
<td>princeton_cadre</td>
<td>189840</td>
<td>164485</td>
<td>143977</td>
<td>124553</td>
<td>15528</td>
<td>22642.5</td>
<td>48834.6</td>
<td>1351.8</td>
<td>1743.2</td>
<td>48702</td>
<td>92430</td>
<td>322</td>
</tr>
<tr>
<td>3</td>
<td>princeton_cadre</td>
<td>110762</td>
<td>134623</td>
<td>116427</td>
<td>118014</td>
<td>15272</td>
<td>23504.8</td>
<td>44321.4</td>
<td>2680.2</td>
<td>1824.3</td>
<td>108188</td>
<td>183333</td>
<td>322</td>
</tr>
<tr>
<td>4</td>
<td>neth_inet</td>
<td>31864</td>
<td>66365</td>
<td>39442</td>
<td>66690</td>
<td>2109</td>
<td>4802.4</td>
<td>9543.5</td>
<td>1943.8</td>
<td>1994.6</td>
<td>9456</td>
<td>2956</td>
<td>243</td>
</tr>
<tr>
<td>5</td>
<td>neth_inet</td>
<td>37577</td>
<td>40815</td>
<td>38263</td>
<td>40288</td>
<td>13524</td>
<td>21242.0</td>
<td>52085.2</td>
<td>921.1</td>
<td>1667.8</td>
<td>1261</td>
<td>13393</td>
<td>322</td>
</tr>
<tr>
<td>6</td>
<td>neth_inet</td>
<td>43223</td>
<td>43314</td>
<td>20288</td>
<td>1994.5</td>
<td>3581</td>
<td>2225.3</td>
<td>5252.0</td>
<td>833.7</td>
<td>186.7</td>
<td>245</td>
<td>7967</td>
<td>143</td>
</tr>
<tr>
<td>7</td>
<td>neth_inet</td>
<td>48413</td>
<td>40402</td>
<td>5155</td>
<td>5222</td>
<td>2417</td>
<td>7554.3</td>
<td>10518.1</td>
<td>238.3</td>
<td>164.3</td>
<td>0</td>
<td>13568</td>
<td>231</td>
</tr>
<tr>
<td>8</td>
<td>neth_inet</td>
<td>48222</td>
<td>32211</td>
<td>2232</td>
<td>3229</td>
<td>112</td>
<td>807.3</td>
<td>2881.7</td>
<td>39.4</td>
<td>79.6</td>
<td>8</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>neth_inet</td>
<td>6614</td>
<td>3154</td>
<td>8570</td>
<td>4394</td>
<td>51</td>
<td>152.7</td>
<td>3804.5</td>
<td>15.8</td>
<td>12.5</td>
<td>0</td>
<td>336</td>
<td>11</td>
</tr>
</tbody>
</table>

http://joiner.co.princeton.edu/status/plot.cfm
More TreeMap Support
Options

- Commercial systems (e.g., OpenView)
  - Good if you can afford it
  - Possibly steep SNMP support curve

- Grid systems (e.g., Ganglia)
  - Easy to deploy for Grid monitoring
  - Extensible, but framework different for extensions

- Ops monitoring (e.g., Nagios)
  - Event monitoring, versus quantity monitoring
  - Good for failure notifications, etc