Ad Hoc Data: An Opportunity for Domain-Specific Languages

Kathleen Fisher
AT&T Labs Research

Joint work with Anne Rogers and Bob Gruber

Abundance of valuable *ad hoc* data

- Call-detail records (fixed-width binary records)
  - Are these calls typical for *this* customer?
  - Are these two numbers owned by the same person?
- Provisioning data (per-order ASCII event sequences)
  - How long does supplier X take to fulfill orders?
  - How many orders sent to supplier X end up being fulfilled?
- Billing system audits (thousands of Cobol data files)
  - Are we paying appropriate taxes in all jurisdictions?
- Internet data (variable-width bin. packets, ASCII messages)
  - Can this packet cause a buffer overflow?

Technical challenges

- Data analysts vary widely in programming ability.
- Data arrives “as is.”
  - Format determined by data source, not consumers.
  - Documentation is often out-of-date or nonexistent.
  - Some percentage of data is “buggy.”
- Often data sources have high volume.
  - Data may not fit into main memory.
  - Data may contain large number of records *and* “entities of interest.”
  - Processing must detect *relevant* errors and respond in *application-specific* ways.

Why might a language help?

- Languages provide very expressive ways of specifying what to do with data, e.g., SQL, XQuery.
- By providing infrastructure, data-processing languages
  - Enable a broader class of people to manipulate data effectively.
  - Shift focus from "How can I compute the information I want?" to "What information do I want?"
  - Shorten programs: easier to write, read, maintain, and reason about.
  - Facilitate error detection.
- Declarative data-specification languages enable generation of a wide variety of tools for manipulating data.

Overview

- Introduction
- Thesis: domain-specific languages facilitate data processing.
  - Hancock: A domain-specific data-processing language for consuming streams of transaction records to maintain customer *signatures*.
  - PADS: A declarative data-specification language for describing physical data formats.
- Conclusions

Hancock: Support for whole data analysis

Individualized analysis: Signatures

- Anomaly detection: fraud, access arbitrage, etc.
- Classification problems: target marketing, biz/res, etc.

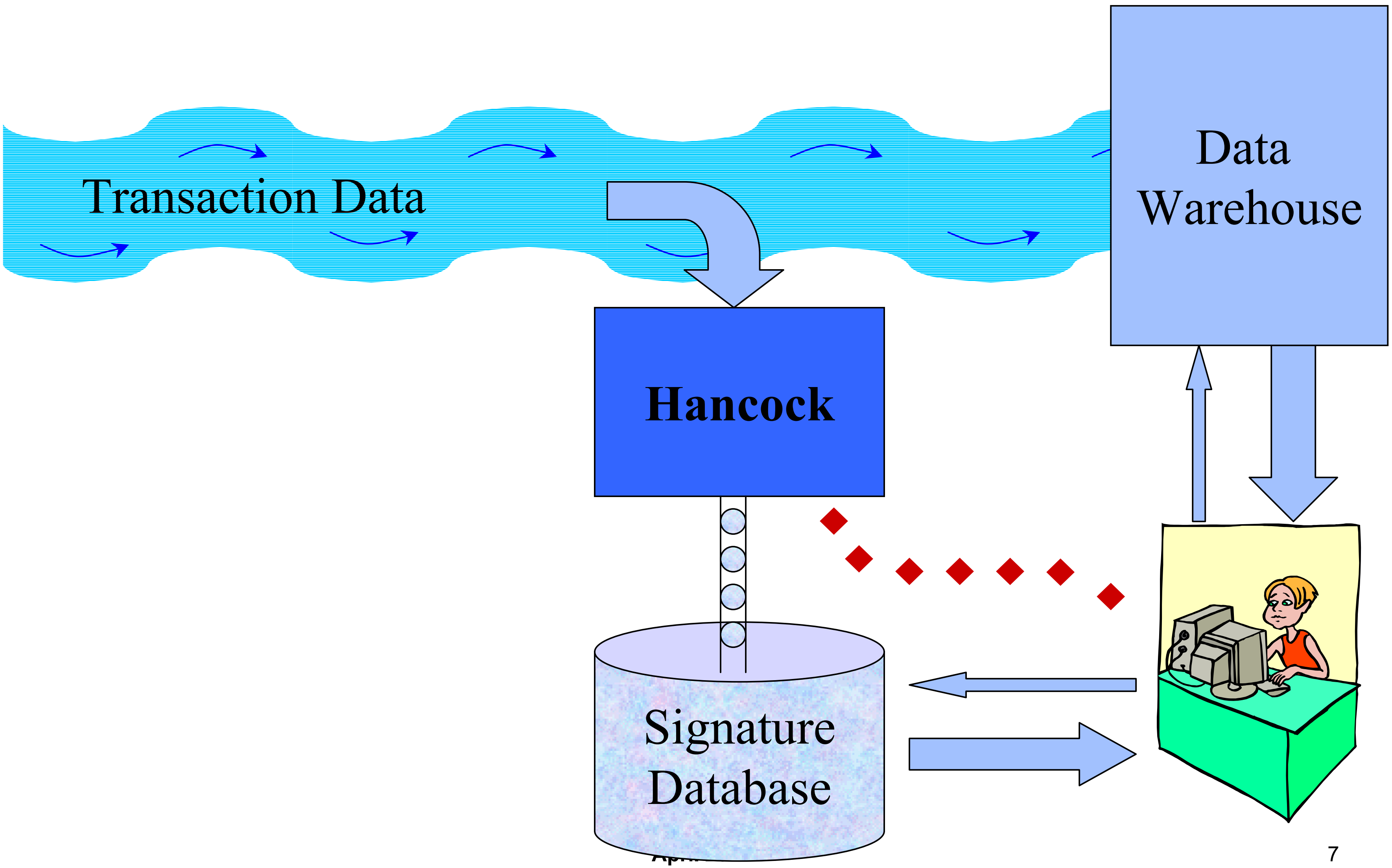
Technical challenge:

- Massive data sets and real-time queries $\Rightarrow$
  Hard I/O and storage requirements $\Rightarrow$
  Complex programs (hard to write, read, and maintain).

Solution:

- A system that reduces the complexity of signature programs.

Processing transactions



Evolution of fraud detection

Country-based thresholds:

- Aggregate calls in 1/4/24 hour windows.
- Compare aggregates to fixed thresholds.
- Exclude common false positives.

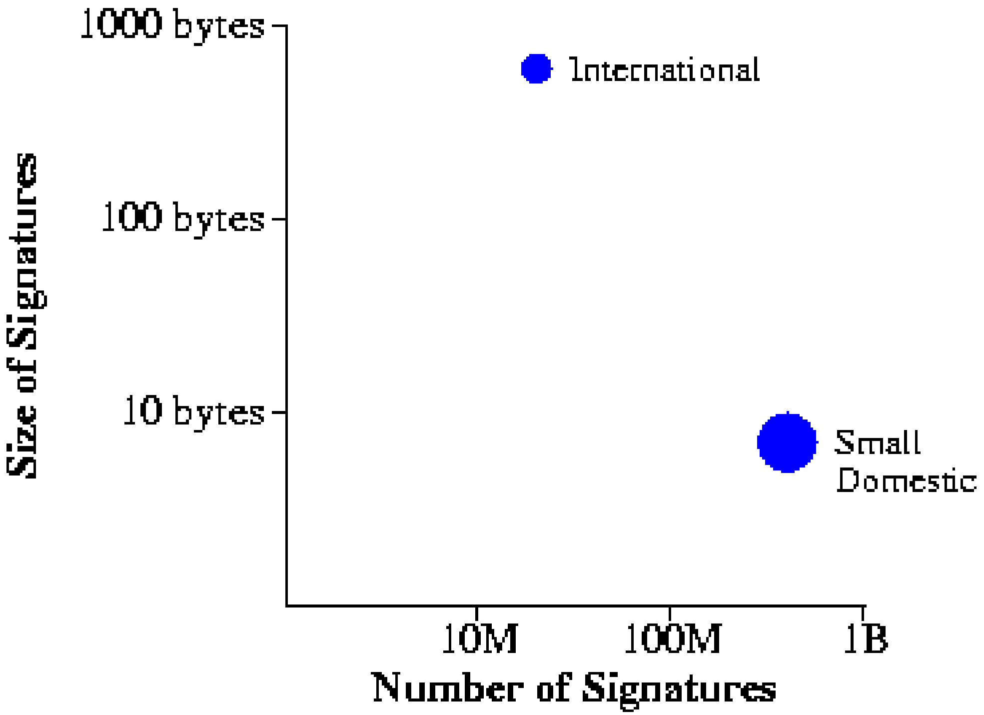
International signatures:

- Signature is an evolving profile.
- Match calls against the customer's and known fraud signatures.

Domestic signatures?

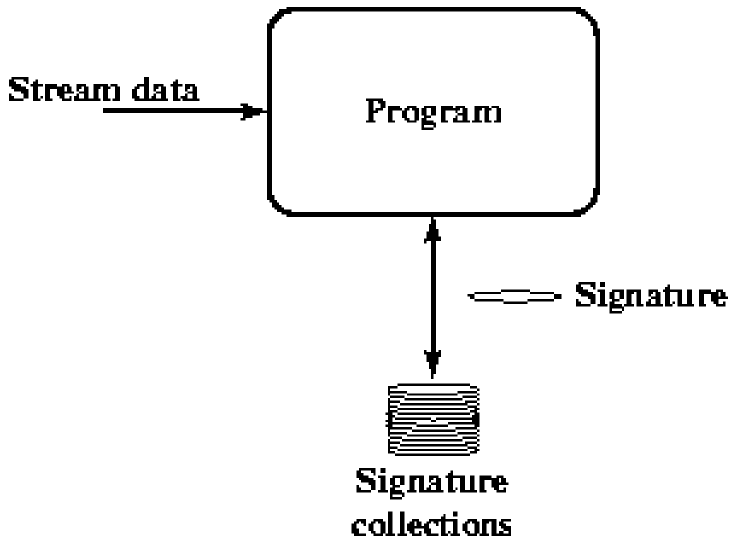
- Much larger scale…

Problem scale



Computational issues

Efficiently managing communications-scale data requires substantial programming expertise.



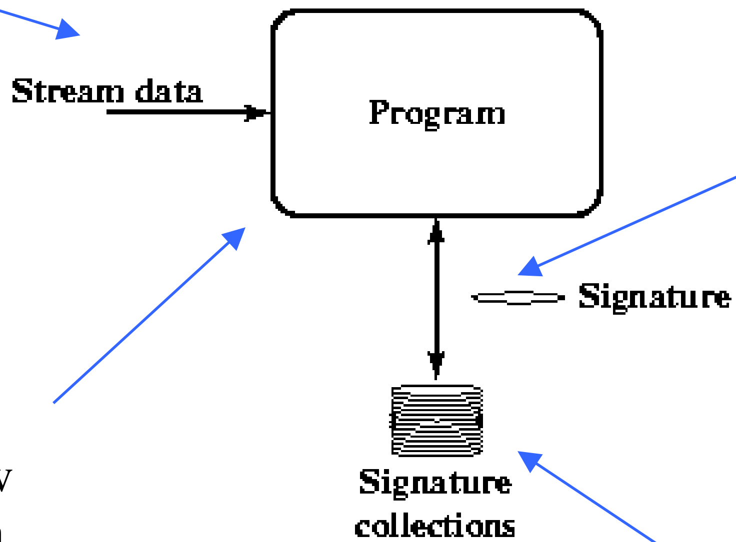
Locality, locality, locality!

Hancock

- Identified abstractions for processing large data streams.
  - Iterated design, meeting with data analysts to get feedback, buy-in.
  - “Wow, you can talk about the things that matter!”
- Embodied these abstractions in Hancock, a C-based domain-specific programming language.
  - Embedding avoided reinventing the wheel, fit user’s comfort zone.
- Built experimental and production signatures using a number of different data streams.
  - All AT&T signature programs are now written in Hancock.

Abstraction overview

Streams. The transactional data to be consumed "daily."



Views. The information to store for each "customer."

Iterate statement. How to combine today's data with historic signatures and other data.

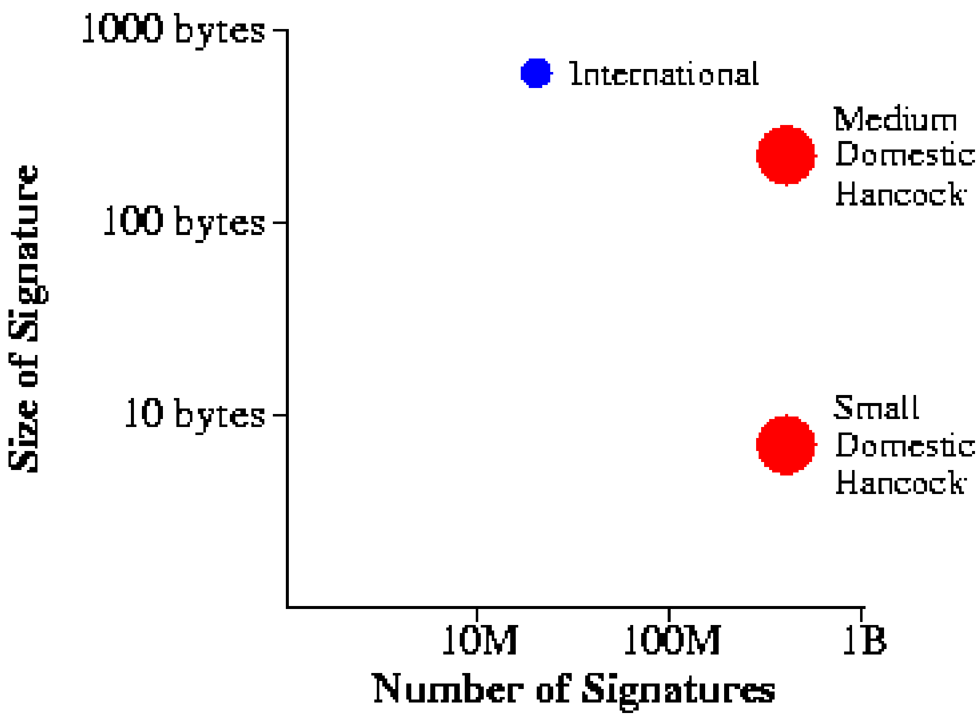
Maps. The collection of customer signatures.

Sample Hancock code

Question: Is this calling behavior normal for *this* number?
Approach: Compute evolving signature for each number to capture normal behavior.

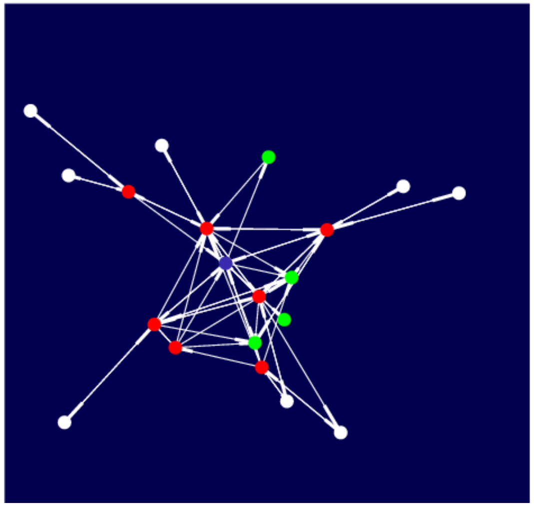
```
iterate
  over        calls
  filteredby  isInternational
  sortedby    origin
  withevents  detectCalls
{
  event line_begin(pn_t pn){ numToday = 0; }
  event call(callRec_t c)  { numToday++; }
  event line_end(pn_t pn)  {
        numCalls<:pn:> = 0.8 * numCalls<:pn:>
                       + 0.2 * numToday;
  }
}
```

Concrete results



Communities of Interest

Question: Are two numbers owned by the same person?
Approach: Compare calling circles:



- Core number
- Inbound calls
- Outbound calls

Hancock summary

- Hancock provides infrastructure for signature programs.
  - Language model reifies abstractions described by analysts.
  - *Data analysts* can easily write efficient signature programs.
  - Programs highlight per-customer computation.
  - Program brevity makes them easy to write, read, maintain, and reason about.
- Embedding Hancock in C
  - Avoided having to design an entire language from scratch.
  - Worked within comfort zone of users.
- Paper: March 2004 issue of TOPLAS

Overview

- Introduction
- Thesis: domain-specific languages facilitate data processing.
  - Hancock: A domain-specific data-processing language for consuming streams of transaction records to maintain customer *signatures*.
  - PADS: A declarative data-specification language for describing physical data formats.
- Conclusions

PADS System (In Progress)

One person writes declarative description of data source:

- Physical format information
- Semantic constraints

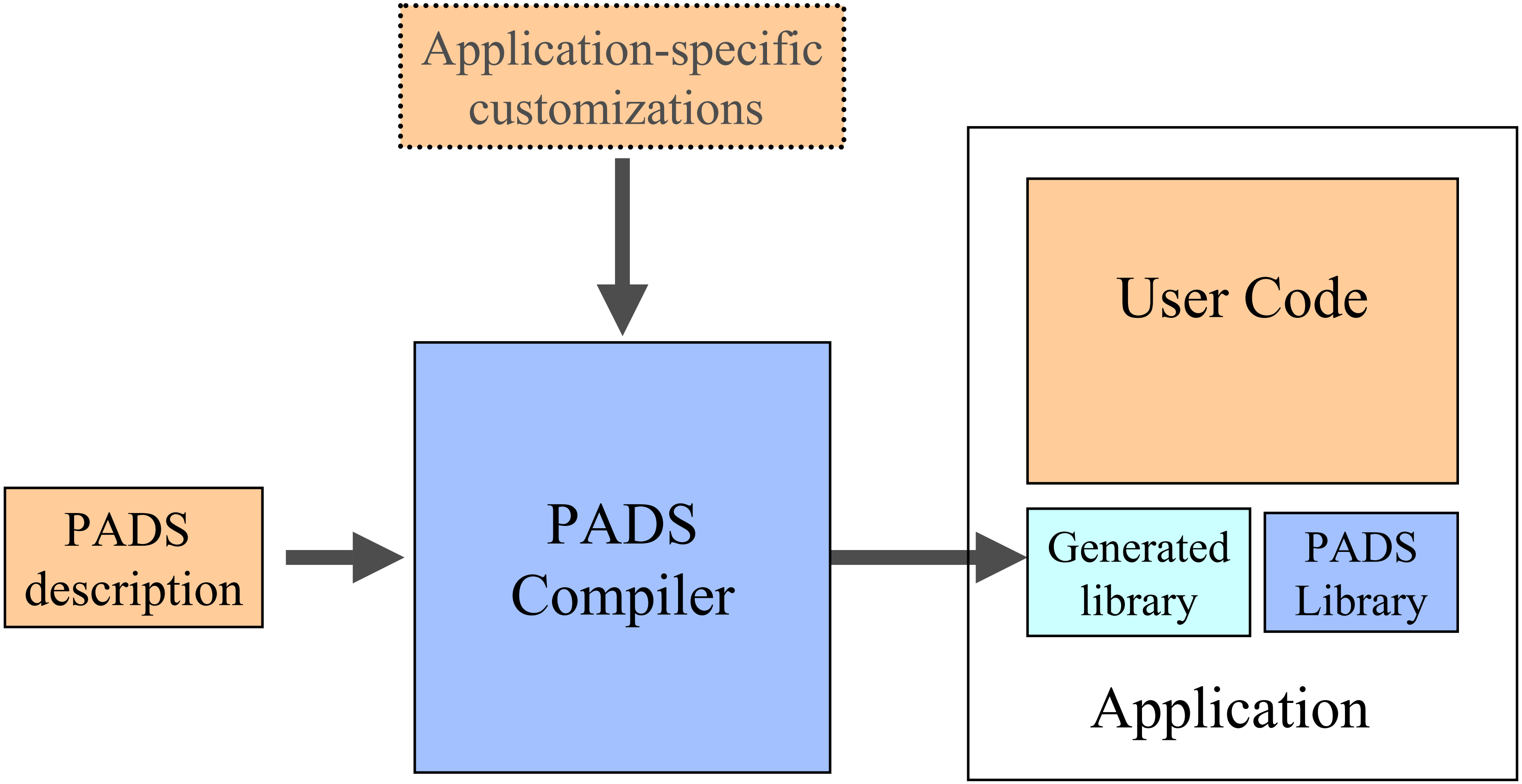
Many people use description and generated library.

- Description serves as maintainable documentation.
- Semantic constraints allow library to detect data errors.
- Bonus: From declarative specification, we can generate (many) auxiliary tools.

PADS applications

- Facilitating Hancock stream descriptions
- Helping statisticians analyze telecom provisioning data
  - Replacing brittle awk/perl scripts
- Auditing billing systems
  - Automatically converting Cobol data
- Analyzing internet packets for conformance to specification
- Loading data into database/stream management systems?

PADS architecture



PADS language

- Can describe ASCII, EBCDIC (Cobol) , binary, and mixed data formats.
- Type-based model: each type indicates how to process associated data.
  - Provides rich and *extensible* set of base types.
    - `Pa_int8`, `Pa_uint8`, `Pb_int8`, `Pb_uint8`, `Pint8`, `Puint8`
    - `Pstring(:`*`term-char`*`:)`, `Pstring_FW(:`*`size`*`:)`, `Pstring_ME(:`*`reg_exp`*`:)`
  - Supports user-defined compound types to describe data source structure:

    `Pstruct, Parray, Punion, Ptypedef, Penum`
  - Allows arbitrary boolean constraint expressions to describe expected properties of data.

Simple example: CLF web log

- Common Log Format from *Web Protocols and Practice.*

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

- Fields:
  - IP address of remote host
  - Remote identity (usually '-' to indicate name not collected)
  - Authenticated user (usually '-' to indicate name not collected)
  - Time associated with request
  - Request (request method, request-uri, and protocol version)
  - Response code
  - Content length

Example: Parray

```
Parray host {
  Puint8[4]: Psep('.') && Pterm(' ');
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Array declarations allow the user to specify:

- Size (fixed, lower-bounded, upper-bounded, unbounded)
- **Psep**, **Pterm**, and termination predicates
- Constraints over sequence of array elements

Array terminates upon exhausting EOF/EOR, reaching terminator, reaching maximum size, or satisfying termination predicate.

Example: Pstruct

```
Precord Pstruct http_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/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

PADS compiler

- Converts description to C header and implementation files.
- For each built-in/user-defined type:
  - Functions (read, write, initialize, cleanup, copy, …)
  - In-memory representation
  - Mask (check constraints, set representation, suppress printing)
  - Parse descriptor
- Reading invariant: If mask is `check` and `set` and parse descriptor reports no errors, then in-memory representation satisfies all constraints in data description.

Example: Reading CLF web log

```
PDC_t *pdc;
http_weblog    entry;
http_weblog_m  mask;
http_weblog_pd pd;

P_open(&pdc, 0 /* PADS disc */, 0 /* PADS IO disc */);
P_IO_fopen(pdc, fileName);
... call init functions ...
http_weblog_mask(&mask, PCheck & PSet);
while (!P_IO_at_EOF(pdc)) {
   http_weblog_read(pdc, &mask, &pd, &entry);
   if (pd.nerr != 0) { ... Error handling ... }
   ... Process/query entry ...
};
... call cleanup functions ...
P_IO_fclose(pdc);
P_close(pdc);
```

PADS: Value-added tools

- Accumulators collect "bird's eye" view of data source (per field percentage of errors, histogram of "Top N")
  - Billing audit: which feeds are interesting/changing/buggy?
  - CLF data: book specification is wrong.
- Interface with Galax implementation of XQuery
  - From PADS description, generate instance of Galax data API.
  - Provisioning data: questions expressible as XQueries (without translating data into XML).
- Canonical translation into XML, including XSchema.
- Web-based data selection programs/general data browser.
- In-memory representation "completion" functions.
- Sanitized test data generation.

PADS: To do list

- Finalize initial release: documentation and release process.
- Conduct careful performance study and tune library accordingly.
- Leverage semantic information to build value-added tools.
- Allow library generation to be customized with application-specific information:
  - Repair errors, ignore fields, customize in-memory representation, *etc*.

Related work

- ASN.1, ASDL
  - Describe logical representation, generate physical.
- DataScript [Back: CGSE 2002] & PacketTypes [McCann & Chandra: SIGCOMM 2000]
  - Binary only
  - Stop on first error
- Database vendors have tools to load specific formats.
- YACC, etc.
- Hand-written parsers in C, perl, etc.

PADS summary

- Data analysts vary widely in programming ability.
  - PADS supports declarative programming, automatic tool generation.
- Data arrives "as is."
  - Format determined by data source, not consumers.
    - PADS language allows consumers to describe data as it is.
  - Documentation is often out-of-date or nonexistent.
    - PADS description can serve as documentation for data source.
  - Some percentage of data is "buggy."
    - Constraints allow consumers to express checked expectations about data.
- Often streams have high volume.
  - Data may not fit into main memory.
    - Multiple entry-points allow different levels of granularity.
  - Processing must detect *relevant* errors (without necessarily halting program)
    - Masks specify relevancy; returned descriptors characterize errors.

Domain Specific Languages

- Facilitate data processing
- Domain-specific abstractions
  - Enable broader class of users to manipulate data.
  - Shorten and simplify user code, improving maintainability.
  - Facilitate error detection.
  - Enable many useful tools.
- Different paradigms useful
  - Procedural vs. declarative
  - Application vs. library
- Embedded DSLs
  - Allow fast prototyping.
  - Avoid duplicating existing functionality.
  - May impede analysis longer term.
  - Can facilitate acceptance by users.

Users are key

- Involve users from the beginning.
  - They understand the domain, its constraints, and what functionality they require.
  - Invaluable input when evaluating the many trade-offs that arise in designing a language.
  - Language will only be successful *if they use it*, so getting them to “buy-in” early is crucial.

Summary

- Hancock (Available online from www.research.att.com/projects/hancock):
  - Domain-specific language for processing arbitrary streams of fixed-width data.
- PADS (In progress. Some information: www.research.att.com/projects/pads):
  - Declarative description of data source, including both layout information and semantic constraints.
  - Compiler generates data-manipulation library.
  - In progress: Suite of tools to leverage declarative specification.

Why not use C / Perl / Shell scripts… ?

- Writing hand-coded parsers is time consuming & error prone.
- Reading them a few months later is difficult.
- Maintaining them in the face of even small format changes can be difficult.
- Programs break in subtle and machine-specific ways (endien-ness, word-sizes).
- Such programs are often incomplete, particularly with respect to errors.

Why not use traditional parsers?

- Specifying a lexer and parser separately can be a barrier.
- Need to handle data-dependent parsing.
- Need more flexible error processing.
- Need support for multiple-entry points.

Getting PADS

PADS will be available shortly for download with a non-commercial-use license.

`http://www.research.att.com/projects/pads`

Example: arrays and unions

```
Parray nIP {
  Puint8 [4] : Psep('.');
};

Parray sIP {
  Pstring_SE(:"[. ]":) [] : Psep('.') && Pterm ('.');
}

Punion host  {
  nIP resolved;  /- 135.207.23.32
  sIP symbolic;  /- www.research.att.com
};

Punion auth_id {
  Pchar unauthorized : unauthorized == '-';
                 /- non-authenticated http session
  Pstring(:' ':) id;
                 /- login supplied during authentication
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Generated type declarations

```
typedef struct {
  host client;          /* Client requesting service */
  auth_id remoteID;     /* Remote identity */
  …
} http_weblog;

typedef struct {
  host_m client;
  auth_id_m remoteID;
  …
} http_weblog_m;

typedef struct {
  int nerr;
  int errCode;
  PDC_loc loc;
  int panic;
  host_pd client;
  auth_id_pd remoteID;
  …;
} http_weblog_pd;
```

Generated accumulator representation (acc)

```
pstruct http_request {
  '\"';  http_method     meth;        /- Request method
  ' ';   Pstring(:' ':)  req_uri;     /- Requested uri.
  ' ';   http_version    version : check(version, meth);
  '\"';
};
```

```
typedef struct {
  PDC_uint64_acc     nerr;
  http_method_acc    meth;
  PDC_string_acc     req_uri;
  http_version_acc   version;
} http_request_acc;
```

Generated read function

```
pstruct http_request {
  '\"';  http_method      meth;        /- Request method
  ' ';   Pstring(:' ':)   req_uri;     /- Requested uri.
  ' ';   http_version     version : check(version, meth);
  '\"';
};
```

```
PDC_error_t http_request_read(PDC_t              *pdc,
                              http_request_m     *m,
                              http_request_pd    *pd,
                              http_request       *rep);
```

We also generate initialization and cleanup functions for representations and error descriptors for variable width data.

Example: *Punion*

```
Punion id {
  Pchar unavailable : unavailable == '-';
  Pstring(:' ':) id;
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

- Union declarations allow the user to describe variations.
- Implementation tries branches in order.
- Stops when it finds a branch whose constraints are all true.
- Switched version branches on supplied tag.

Advanced features: User constraints

```
int checkVersion(http_v version, method_t meth) {
  if ((version.major == 1) && (version.minor == 0)) return 1;
  if ((meth  == LINK) || (meth == UNLINK)) return 0;
  return 1;
}

Pstruct http_request {
  '\"'; method_t        meth;     /- Request method
  ' ';  Pstring(:' ':) req_uri; /- Requested uri.
  ' ';  http_v          version : checkVersion(version, meth);
                               /- HTTP version number of request
  '\"';
};
```

207.136.97.50 - - [15/Oct/1997:18:46:51 -0700] "GET /turkey/amnty1.gif HTTP/1.0" 200 3013

Advanced features: Sharing information

- "Early" data often affects parsing of later data:
  - Lengths of sequences
  - Branches of switched unions
- To accommodate this usage, we allow PADS types to be parameterized:

```
Punion packets_t (: Puint8 which, Puint8 length:) {
  Pswitch (which) {
    Pcase  1: header_t  header;
    Pcase  2: body_t    body;
    Pcase  3: trailer_t trailer;
    Pdefault: Pstring_FW(: length :) unknown;
};
```

Generated representation

```
Pstruct http_request {
  '\"';  http_method     meth;         /- Request method
  ' ';   Pstring(:' ':)  req_uri;      /- Requested uri.
  ' ';   http_version    version : check(version, meth);
  '\"';
};
```

```
typedef struct {
  http_method    meth;         /* Request method */
  Pstring        req_uri;      /* Requested uri */
  http_version   version;      /* check(version, meth) */
} http_request;
```

Generated mask (m)

```
Pstruct http_request {
  '\"';  http_method      meth;        /- Request method
  ' ';   Pstring(:' ':)   req_uri;     /- Requested uri.
  ' ';   http_version     version : check(version, meth);
  '\"';
};
```

```
typedef struct {
  P_base_m         structLevel;
  http_method_m    meth;
  P_base_m         req_uri;      /* Check, Set, Print,… */
  http_version_m   version;
} http_request_m;
```

Generated parse descriptor (pd)

```
Pstruct http_request {
  '\"';  http_method      meth;          /- Request method
  ' ';   Pstring(:' ':)   req_uri;       /- Requested uri.
  ' ';   http_version     version : check(version, meth);
  '\"';
};
```

```
typedef struct {
  int                 nerr;
  P_errCode_t         errCode;
  P_loc_t             loc;
  int                 panic;         /* Structural error */
  http_method_pd      meth;
  P_base_pd           req_ uri;
  http_version_pd     version;
} http_request_pd;
```

Expanding our horizons

- Design to this point primarily based on our experiences with Hancock data streams.
- To get feedback on the expressiveness of our language and the utility of the generated libraries, we started collecting other users:
  - Data analysts (Chris Volinksy, David Poole)
  - Cobol gurus (Andrew Hume, Bethany Robinson, …)
  - Internet miner (Trevor Jim)

Data analysts: The domain

- Data: ASCII files, several gigabytes in size, sequence of provisioning records, each of which is a sequence of state, time-stamp pairs.

```
customer_id | order# | state1 | ts1 | … | staten | tsn
```

- Desired application: aggregation queries.
  - How many records that go through state 3 end in state 5?
  - What is the average length of time spent in state 4?
  - …
- Original applications written in brittle awk and perl code.

Data analysts: Lessons learned

- PADS expressive enough to describe data sources.
- PADS descriptions much less brittle than originals.
- Generated code faster than original implementations.
- Support for declarative querying could be a big win, as analysts can then generate desired aggregates with only declarative programming.
  - Data and desired queries “semi-structured” rather than relational.

Developing support for declarative querying

- Wanted to leverage existing query language.
- XQuery, standardized query language for XML, is appropriately expressive for our queries.
- Modified Galax, open-source implementation of XQuery, to create data API, allowing Galax to “read” data not in XML format. (Mary Fernandez, Jérôme Simeon).
- Extended PADSC to generate data API.
- Currently, can run queries over PADS data if data fits in memory.
- Next: extend Galax API and generated library to support streaming interface.

Cobol gurus: The domain

- Data: EBCDIC encoded files with Cobol copy-book descriptions. Thousands of files arriving on daily basis, hundreds of relevant copy-books, with more formats arriving regularly.
- Desired application: developing a “bird’s eye” view of data.
- Original applications: hand-written summary programs on an *ad hoc* basis.

Cobol gurus: Lessons learned

- Wrote a translator that converts from Cobol copy books to PADSL.
  - Added **`Palternates`**, which parse a block of data multiple times, making all parses available in memory.
  - Generated description uses **`Palternates`**, **`Parrays`**, **`Pstructs`**, and **`Punions`**.
  - Uses parameters to control data-dependent array lengths.
  - Successfully translated available copybooks.
- Added *accumulators* to generated library.
  - Aggregate parsed data, including errors.
  - Generate reports, providing bird’s eye view.

Internet miner: The domain

- Data: data formats described by RFCs. Some binary (dns, for example), some ASCII (http, for example).
- Desired applications: detecting security-related protocol violations, data mining, semi-automatic generation of reference implementations.
- Original applications: being developed with PADS.

Internet miners: Lessons learned

- Constraints are a big win, because they allow semantic conditions to be checked.
- Parameterization used a lot, particularly to check buffer lengths in binary formats (dns).
- Additional features needed:
  - Predicates over parsed prefixes to express array termination.
  - Regular expression literals and user-defined character classes.
  - Positional information in constraint language.
  - Recursive declarations.
  - In-line declarations.
- Developing script to semi-automate translation of RFC EBNF (Trevor Jim).
  - URI spec successfully translated, HTTP close.