Notes on DSL Design for Email Transaction Processing

Draft Working Paper

Michal Young
CIS 410/510 DSL

Draft Version of 3 November 2000

Abstract

This document is a set of working notes on the design of a programmable system for processing transaction requests carried in electronic mail messages. A "transaction" in this sense could be a homework assignment, a take-home exam, or a conference paper review.

Contents

1 Background

We have tentatively concluded that electronic transaction processing over an electronic mail connection is a domain that could be profitably supported by a domain-specific language, and that designing and implementing such a language may be a sufficiently small task for the Fall 2000 CIS 510/410 class to take on as an exercise or project.

2 Basic Requirements / Functionality

We identified the following as potential functionality for processing of transactions in electronic mail:

Extract: Extract transaction data from email messages. This could include both content and "envelope," or metadata. (An example of metadata is the size of a piece of data, or its content-encoding in the MIME sense.)

- Both message body text and MIME enclosures.
- Message headers as well as message body.
- Possibly some "virtual" data that is not actually present in the message, such as the current time when it is being processed.

It may be useful (or necessary) to distinguish between short text fields from the message body, and larger items that should be stored as files.

Validation: It should be easy to validate transaction data. This includes ensuring that all required data is present, and that it conforms to certain rules. For example, a multiple-choice exam question might have possible answers "A," "B," "C," or "D," but not "X" or "12." When validation fails, it should be possible to send a diagnostic message, possibly produce a log record, and then stop without proceeding further in processing.

Possible validation criteria include:

- Presence (the field is present in the form).
- Non-empty (the field is not only present, but has a non-null value).
- Length or size (the length of the field falls within a specified range). This could be a length in characters, in words, in lines (at some specified line length?), or possibly something else.
- Element of enumeration (like the multiple-choice example above). This may be distinct from looking up items in an external database or table.
- Textual pattern match (e.g., a regular expression).
- Presence in external table or file.
- Common syntactic types: email address, URL, identifier, possibly others.

- Validation by external function or program.

It is unlikely that all of the possible validation criteria would be supported. Some combinations make others superfluous. For example, if there is a facility for matching against a regular expression, and a facility for checking an external table or file, then the "element of enumeration" check is redundant.

Access control: Access control includes authorization (e.g., is this really a student in the class?) and sandboxing. Both seem to be essential, although sandboxing may not need to be very sophisticated. Access control might be subsumed by validation (specifically, by looking up combinations of fields in an external table), although it might impose some additional requirements (cryptographic signatures?).

Filtering: There should almost certainly be a facility to feed form data to an external program or function[1]

Response: Email transaction processing always involves generation of email responses. It must be possible (and easy) to construct responses that include both fixed text and some data extracted from messages or returned from validation operation and external functions or programs.

Logging, Storage: It must be possible to store both the raw data from transaction processing, and possibly the output of validation and/or filtering as well. It is likely that "logging" output (individual records consisting of several fields pertaining to one email transaction, accumulating in one file or database) will be separate from larger-grain storage that produces several files (e.g., program text sent for grading).

There is more than one possible storage policy for storage and logging. Possibilities include:

- Cumulative: Make a new record or set of files for each transaction, even if several transactions are received with the same identifying information.
- First-only: Store only the first-received data. Reject further data.
- Last-only: After the first transaction, subsequent transactions with the same identification replace earlier versions.

Regardless of storage policy, validation should be able to prevent accepting transactions that arrive after a deadline, using pseudo-data such as the current date and time.

Addressing: This follows from a decision to minimize the amount of procmail programming that a user should be required to do. If the user is not using procmail to route each mail message to the appropriate processing script, then the scripting language will have to take over that functionality.

[1] An "external function" means one outside the DSL. It could, for example, be a function written in the host language, like an action routine in YACC.

3 Design Overview

3.1 Desiderata

My design goals include:

- Current and familiar technology base. What this means, basically, is that people should be able to prepare email messages to turn in homework, reviews, etc., using their standard software, without acquiring new tools or learning much. This may sound obvious, but it was actually a tough choice — requiring email messages to be in XML format would make things much easier, and would add some capabilities that are absent from my current design.
- Separation of email form encoding from processing instructions. This is partly in response to the previous point. I believe that in the future, XML will be the right and obvious way to send transaction data in email messages, and there will be ample tool support for preparing email messages that contain XML data. Therefore, while depending on XML at this time would violate the first design goal, the email encoding should be well-enough separated from other aspects of system and language design that it would be simple to switch to an XML-based system later.
- Simplicity, brevity, etc. — the usual stuff. Where forced to choose between the facilities that I would ideally like, and those that I think I know how to implement simply, I have chosen simple implementation.

3.2 Decision: Fixed form language

I considered three main alternatives:

Flexible template language: Provide the user (the person defining form content) flexible facilities for describing what an email message should look like and how transaction data in an email message will be recognized.

Fixed form language: Impose fixed rules for what an email message should look like and how transaction data in an email message will be recognized.

XML message format: Require that email transactions be in XML format. This is a variation on requiring a fixed form language, but with special considerations. XML is human-readable and almost human-writeable if sufficiently constrained. XML document-type-definitions or schemata could be used to specify much of the simple validation criteria, which would then be processed by off-the-shelf tools.

I was sorely tempted by the XML alternative. XML is rapidly becoming a standard in many kinds of data interchange. Good, free tools are widely available for processing XML, and they include tools for validating an XML document against a *document type description* (DTD) or schema. I believe

that in three years or less, choosing XML will be a no-brainer. Nonetheless I rejected it at this time for the following reasons:

- Even though XML processing tools are widely available, most people sending email messages as XML would probably not have appropriate tools installed, or would not be familiar with them.
- Few users would be able to construct DTDs or schemata to define acceptable content. Moreover, the messages produced by current XML validators would not be suitable error messages to enclose in responses.
- Although in principle data in other formats (e.g., MP3 files or Java programs) can be "encapsulated" in XML, in practice the simple tools that are widely available now assume that XML documents contain only simple text.

Having reluctantly rejected the XML alternative, it seemed that the best approach was to adopt a simple, fixed format as a stop-gap until XML usage is more mature and better supported. This will most likely take the form commonly seen in human-processed email forms, e.g.,

> field label: field content

for data that fit on one line and

> field label:
> content content content content
> content content content content

for data that may span several lines.

4 The Scripting Language

At this point it is easier to discuss ideas about the scripting language through a running example.

4.1 Check blocks

One of the fundamental constructs of the language as I currently imagine it is "check blocks." Although they look something like "try" blocks with exception handlers in Java, their purpose is almost the opposite. Whereas a "try" block allows any statement to immediately jump to the exception handler, a "check" block is executed in its entirety, whether individual checks succeed or fail. Any error messages produced during execution of the "check" block are saved. Then, if any of the checks have failed, a "failure" block is executed; this is where the error messages may be communicated to the user. The program halts after execution of the "failure" block.

```
logfile = "/home/michal/classes/cis510/log" ;
check {
  field surname [Last name] text(1) required;
  field givenname [First name] text(1) required;
  field homepage [Home page URL] text(1) optional;
} failure {
  reply <<
Your message cannot be processed because
some required fields were missing or were not
recognized:
$(messages)
>>
}
```

In the fragment above, the "field" statements describe data that may be extracted from a message. The identifier used to refer to this data within the script may not be the same as the label used to identify it in the email message. For example, the variable `surname` might have appeared in the email message as

```
Last name:  Jones
```

4.2 Field Types

Part of the validation is given by associating a type with each field. In the example above, `text(1)` accepts a single line of text, which should occur on the same line as the label. Other built-in types might include

text(*n*) Up to *n* lines of text. (Note: it is not clear that "lines" is the right way to define maximum size.)

int An integer. It might also be useful to specify an acceptable range.

url A universal resource locator (URL). Note that valid URLs include not only `http:` scheme resources, but also `ftp:` and `mailto:` scheme resources. In most cases it may be better to indicate specifically the kind of URL that is required.

It is probably useful to keep the type system "open" in the sense that new types can be defined along with their validation routines (written in the host programming language). However, there is a tradeoff — an open type system probably precludes using the type as an aid to actually extracting the fields.

4.3 Email Replies

The example above shows a possible syntax for generating email messages and including the content of program variables (in this case, the built-in variable `messages` which contains all the error messages generated during execution of a `check` block). However, this example does not specify anything about

the email headers. The advantage of not specifying email headers is that the user cannot accidentally generate bogus headers. The disadvantage is that the user also cannot provide headers that are more meaningful than whatever defaults the interpreter generates. For example, the interpreter could probably fill in a subject like

```
Failure notice re: original subject
```

but the user might supply a much more meaningful subject line like

```
Your submission of homework 3 could not be processed
```

It might be enough to treat any line in a `reply` command that begins with an RFC-822 style label as a header, providing default headers when explicit headers are not given.

4.4 Simple Example

Here is a slightly more complete example:

```
basedir = "/home/michal/classes/cis510" ;
logfile = basedir + "/log" ;
check {
  field surname [Last name] text(1) required;
  field givenname [First name] text(1) required;
  field homepage [Home page URL] text(1) optional;
} failure {
  log logfile {
     email now "Identifying information missing"
  }
  reply <<
Subject: Unable to process your homework submission
Your message cannot be processed because
some required fields were missing or were not
recognized:
$(messages)
>>
}

check {
   field q1 [Answer Q1] text(1) required;
   field q2 [Answer Q2] text(1) required;
   field q3 [Answer Q3] text(30) required;
   field e1 [Answer E1] text(30) optional;
} failure {
  log logfile {
      email
      now
      surname + ", " + givenname
```

```
      "Unable to parse some answers"
  reply <<
Subject: Unable to process your exam
Some of your answers could not be processed. Please
try again.
$(messages)
>>
}
thedir = basedir + unique(surname + givenname + now);
mkdir thedir;
store thedir + "/answers/q1" q1;
store thedir + "/answers/q2" q2;
store thedir + "/answers/q3" q3;
store thedir + "/answers/q4" q4;

reply <<
Subject: Your exam has been received
A message with your homework has been received and stored
for human grading.
```

4.5 TBD

Things that need to be addressed but are not addressed in the example above include: authentication, storage policies, storage formats (comma-separated values, XML, external databases), filtering, general validation against patterns or tables. There are probably more omissions.

5 Working Example 1: Database Homework

Kevin Redwine's example produces a directory for each student and places the content of each answer in a separate file. Here is how it might look using this system.

```
basedir = "/home/michal/classes/cis510" ;
logfile = basedir + "/log" ;
check {
  field surname [Last name] text(1) required;
  field givenname [First name] text(1) required;
} failure {
reply <<
Subject: Unable to process your homework submission
Your database homework cannot be processed because
some required fields were missing or were not
recognized:
$(messages)
>>
}
```

```
check {
   field q1 [1] text required;
   field q2 [2] text required;
   field q3 [3] text required;
} failure {
reply <<
Subject: Unable to process your exam
Some of your answers could not be processed. Please
try again.
$(messages)
>>
}
thedir = basedir + surname;
mkdir thedir;
store thedir + "/answers/q1" q1;
store thedir + "/answers/q2" q2;
store thedir + "/answers/q3" q3;
store thedir + "/answers/q4" q4;

reply <<
Subject: Your homework has been received
A message with your database assignment 1 has
been received and stored for Kevin to grade.
>>
```

6 Working Example 2: ICSE 2001 Review

For conference reviewing, I would probably distribute a "template" message to all program committee members, mixing together instructions and results. The reviewing form is rather long, with a lot of repetition of a few basic patterns, but I believe that it is best to look at the whole thing rather than an excerpt in order to judge issues of brevity and usability. The template of an email message would look something like the following. (Note that I am trying to follow the actual ICSE 2001 reviewing web form very closely "as is," rather than revising it to fit my own ideas of what a review form ought to ask.)

```
Subject: ICSE 2001 Review


======================
Identifying information
======================
Replace _ in the following fields

Paper number: _
Title: _
Authors: _
Reviewer Identification: _
```

```
=======================
      Summary
=======================

Please provide a short summary/overview of the paper in your own
words.  It will be sent to the author.

Summary:
  (Replace this text with your summary.  It may be several lines
  long.)


.  (do not change this line)

========================
   Classification
(Special tracks only)
========================

For papers submitted to the Case Studies or Education track only ---
How would you classify this paper?
Use one of the following classification codes:
EXR = Experience reports  (Case Studies track only)
EXM = Exemplars           (Case Studies track only)
PSE = Ph.D. programs in S.E.               (Education track only)
DE  = Distance education programs in S.E.  (Education track only)
OTH = Other

Classification: _

==============================
          Ratings
==============================

For the following questions, use a digit 1-5 with the following
interpretation:
  1 = high, 2 = medium-high, 3 = medium, 4 = medium-low 5 = low
Replace the _ for each question

Does the paper cleary identify and formulate a
problem/challenge/issue?
Impact: _

Does the paper provide novel solutions/experiences/insight?
Originality: _

Does the paper provide convincing
evidence/evaluation/argumentation?
Correctness: _

Is there sufficient presentation of background and related work?
Technical depth: _

What is the overall quality of the presentation?
Presentation: _

What is the likelihood that presentation problems can/will be fixed in
time?
Fixable: _
```

```
======================
  Summary evaluation
======================
  Use the following scale -
    A = I will champion this paper at the PC meeting.
    B = I can accept this paper, but I will not champion it.
    C = This paper should be rejected, but I will not fight
        strongly against it.
    D = Serious problems.  I will argue to reject this paper.

Summary evaluation: _

====================
  Confidence
====================
Use the following scale
  X = I am an expert
  Y = I am knowledgeable in the area, though not an expert
  Z = I am not an expert.  My evaluation is that of an informed
      outsider.

Confidence: _

====================
  Journal fast track
====================
Should we consider a version of this paper for journal publication?
Journal: _
  Y = yes, N = no

==================
  Comments
==================

The comments in these fields may as long as you like.

For the program committee only --
Private comments:
  (Replace this with your comments, if any)

.do not change this line

For the authors --

Positive aspects:
  (Replace this with your comments on positive aspects
  of the paper, which will be sent to the authors)

. do not change this line

Negative aspects:
  (Replace this with your comments on negative aspects
  of the paper, which will be sent to the authors)

. do not change this line

Detailed comments:
  (Replace this with detailed comments, which will be
   sent to the authors)
```

```
. do not change this line
```

6.1 TBD

I need to write the script file to go with this.

7 Simplicity through defaults?

In class discussions, there has been considerable opinion on the side of making the scripting language simpler and making it look less like programming. I am not very concerned about superficial syntactic details like the use of curly braces, but I am concerned about the convenience of writing simple scripts. If the script equivalent of "hello world" is long or complicated, then no one will use the tool. If scripts for simple tasks are very simple, then users will become hooked and learn to use more advanced features as they are needed.

Is it possibly to hide most of the available complexity of scripts through reasonable default behavior? (**TBD:** Finish this section.)

7.1 Defaults

TBD.

7.2 Limitations

Complexity that can't be hidden with defaults. TBD.