ENHANCED ADL

DESIGN DOCUMENT
&
USER GUIDE

Zhigang Yin

CONTENTS

Introduction to current ADL language

"ADL is a language for describing how a data file is structured and how to convert it into an Information Tools formatted database. There are a number of other tools that are used for manipulating data (before running ADL) and for working with Information Tools formatted databases once they have been created (after running ADL). In either case, ADL is a key part of the process because it encodes data into the Information Tools special format."

------ *(ADL V3 Manual)*

There are 25 (or more) people currently been employed to work on a task described as "data setup". This group of employees are called the "setup-team".

Generally speaking, the setup-team's task is that they accept raw data (primarily survey data) in various formats, and then use *ADL* as well as supporting programs from the "*iTools*" [1] suite to produce ITL formatted databases, namely "X-files".

The primary tool of the data set-up team is considered to be the ADL program, as at present that is the only program that can generate the ITL formatted databases for end-users. Other companies also use the ADL program worldwide, to produce databases for client analysis.

Since the ADL program is a fundamental part of the Information Tools business model, my project aims to develop better, more industry standard ways of doing the ITL setup task. This is to be accomplished by programmatically designing either some form of enhancing component for ADL or by re-writing the ADL completely. Different options are explored, and applicability of each alternative is discussed through this document. I hope to improve not only the ADL program itself, but also the production task in which they process data and build databases for customer analysis. This means higher efficiency and quality.

[1] "iTools" suite: a set of in-house programmes used by ITL.

Requirements for "Enhanced ADL"

--Conceptual requirements:

1) **Training:**
   Enhanced ADL should have a shallow learning curve. Least training required for new employees.

2) **Speed:**
   Encompasses the processing time required to build a database
   More importantly, the time required to write a complete script

3) **Correctness:**
   - Correctness in the build process in which the script is turned into a database with no ambiguity.
   - Correctness in the readability or the language and program reporting tools to enable input/output checking.

4) **Easy to maintain and extend to meet new challenges.**
   - Flexibility: capable of dealing with new types of data
   - Integration: can integrate with relational database 'warehouse' tools or applications written in other languages.

5) **Retain backwards and forwards compatibility**.
   ITL has some 10 years or more of data held in ITL database form with ADL descriptions of much of the data. It is therefore vital that any solution can run along side of ADL and still produce ITL databases at least in the medium term.

6) **Extra features:**
   - Context sensitive help
   - Modern language features (e.g. syntax completion).

--Detailed requirements from ADL users:

Error handling:

- "Implement the Error message on duplicate axes. "

- Error message if base applied incorrectly. "The manual states that Base can not be applied to value axes. However, ADL does not stop people from running a base on these items. Therefore, I think we need to do one of two things, either we change ADL to support base on this type of axes, or we change ADL to return and Error if a user attempts to do this.

- Give error message rather than warning message if preset tables are incorrectly set up

Calculation:

- "…, takes a denominator and an enumerator & calculates a probability between 0 and 1…."
- Min and Max functions like Count but return the min and maximum value of a group of locations or variables.

Security:

- Restrict writing auxiliary files such as .ADT, .USR and .DBG files unless required to.

- Make registration like V3 - prompt for details if no registration, automatically attempt to register if .RGS dir is found.

Variable types support:

- Byte and Word keywords. Generate error/warning if multiple response and/or number of elements overflows byte/word axis. ADL should automatically compress data if possible to allow these keywords to be dropped
- Re-design of variable handling.

Other features:

- Reference multiple data files at one. Could assign to variables so:

```
sales_data = "sales.dat" maxrec 5
survey_data = "c:\data\survey.csv" filetype delimited ","

["Axis 1" where=sales_data:2:1_5]
["Axis 2" where=survey_data:1 to 3]
```

- "Few issues of course - like do we default to the last used file until the next reference? Or do we have separate sections of the ADL file – like "begin file1" "end file2" or just use includes?"

- Compound axes. Col macros are used for a list of elements and their column positions. Would like multiple where statements to be used so that you could have multiple macros used.
  For instance:

```
define banks
 "bank1" = 1
 "bank2" = 1
 "bank2" = 1
enddef

 define others
   "not answered"
   "no bank"
 enddef

 ["Like service" col where=100 banks
  where = 150 others
 ]
```

- Add axes keywords: 'media' and 'header'. Possibly add 'PITA' flags?

- Debug off - "Please could we add to the development list a request for debug to also respond to debug off. Other, Suppress and Base all respond to off, while debug is turned off through the command debug 0. It would be good if it also responded to debug off, just for consistency sake!"

- Shrink list for finding axes (elements?). Bookmarking / syntax highlighting editor.

- Spelling checker / abbreviation lookup. Would parse ADL file and list suspect words. Also would be able to search and replace words like ModAxes.

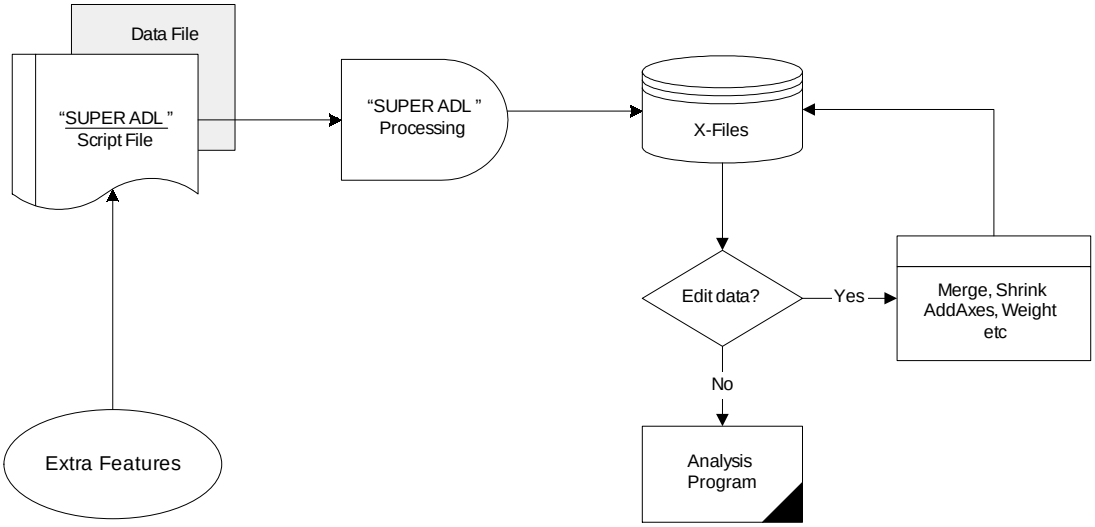
- Concatenate strings (var = string1 + string2)?

- Test incrementing with “col” based macros.

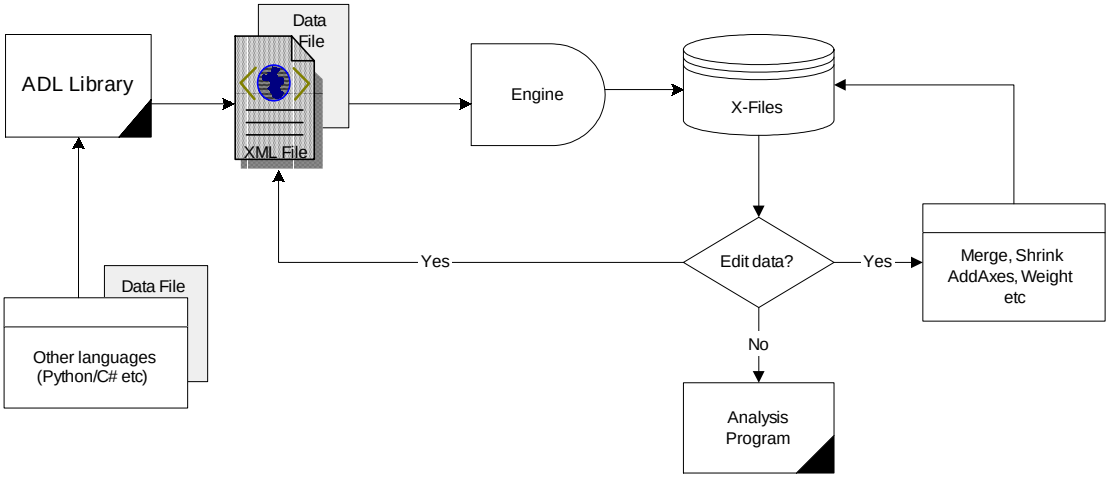
- Meta-axes/elements axes/elements based on the counts generated by other base axes/elements

Initial design models

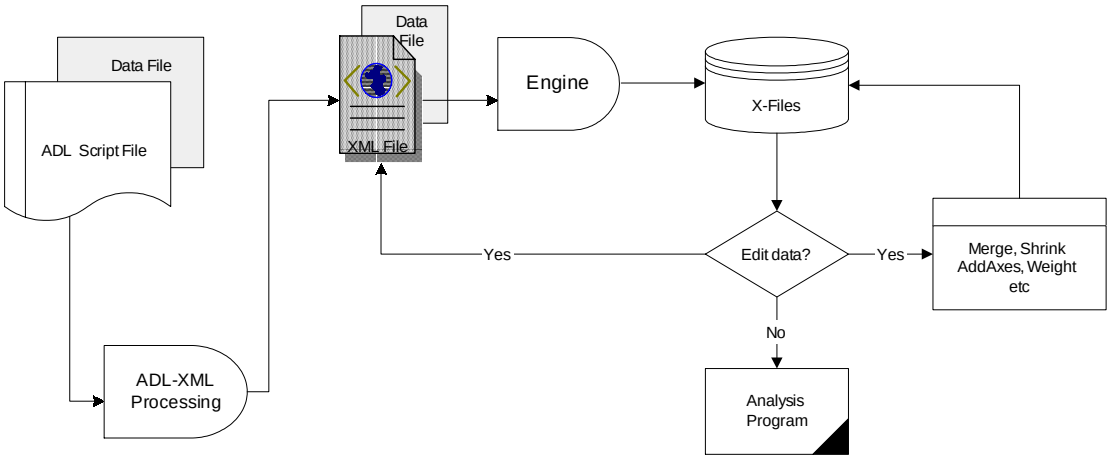
Model One: “SUPER ADL” ---- Original ADL language with extra features.



Model Two: Use another programming language



Model Three: XML format of X-Files Database & XML to X-Files engine



Pros & Cons:

Model	Pros	Cons
One	Minor changes to the original ADL structure, easy training	Not much contribution to future expansion No change to compatibility with other DB formats
Two	Can take the advantage of existing programming language, much richer features supported. e.g. error handling, debugging	Based on deep understanding of the original ADL programme code. This is extremely time-consuming. Deep training curve expected for employees with no programming background.
Three	Formalized database structure. Easy to change, easy to read, easy to generate. Provides good flexibility and compatibility.	Not much improvement on initial set-up process. Requires higher standard of original ADL script file.

Rating:

	Model One	Model Two	Model Three
Speed	2	3	1
Correctness	2	1	3
training /re-training	1	3	2
Flexibility	2	3	3
Integration	2	3	3
Backward compatibility	1	3	2
Forward compatibility	3	2	1
Error handling	3	1	2
Security	2	1	3
Programming workload	1	3	2
Change to original language structure	1	3	2
Complexity of Use	1	3	2

“1”---- Good **“2”----Neutral** **“3”---- Bad**

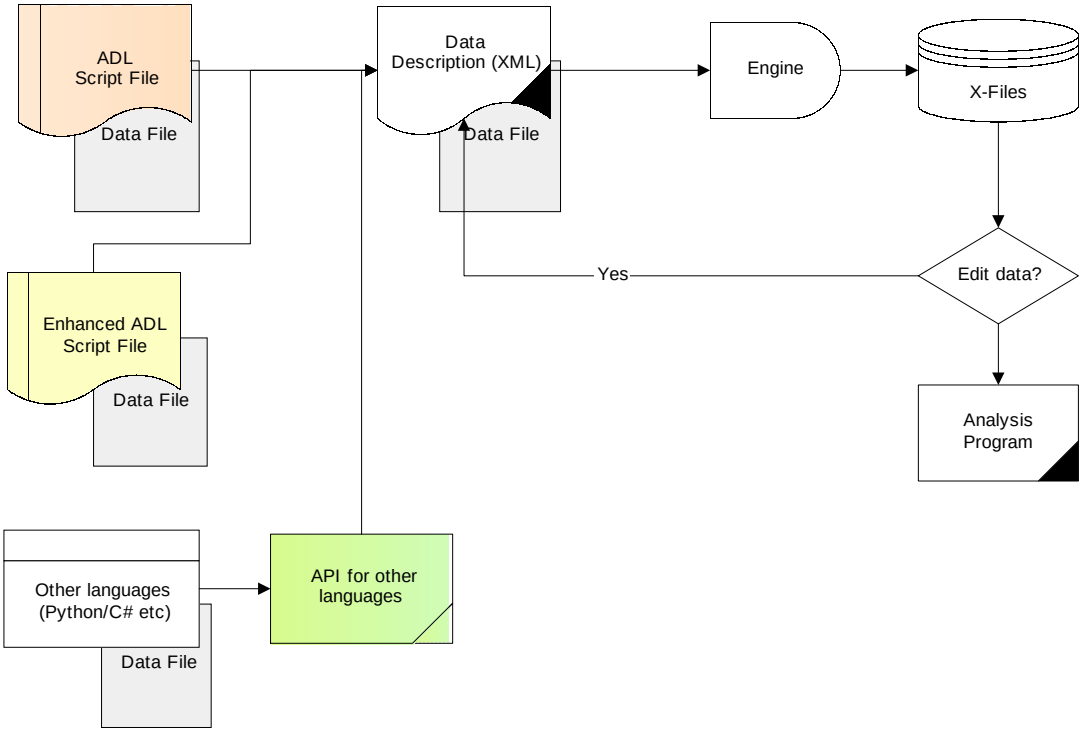
Decision:

From the table above, it’s clear that each model has its own advantage and disadvantage. The more advantages my final solution can have the better. However, this is limited by both the time and the complexity.
My decision for now is Model, i.e. to rewrite ADL in another programming language, using API for the programming language (XML handlers, libraries), and output a XML file as an intermediate format. A convert engine will then

take this XML file and the original .DAT file as input in order to generate the ITL Database in X-Files format.

Ideal Model:

Ideally, I hope to make enhancement to current ADL that includes all the advantages of the three different models and avoids all the disadvantages. That is, script files can be written in either enhanced ADL syntax ("Super ADL") or in an existing programming language. Both generate an XML format data description file. This is then combined with the raw data file, feed through the XML to X-Files convert engine to generate the X-Files Database.
This is possible if time allowed. It's also possible that new projects are set up to further develop the idea programmatically.



Reference:

- Information Tools

  --- "ADL V3 Manual" (latest version)
  --- "ADL Enhancement Proposal" by Grant Black
  --- *http://www.infotools.com*

- "Creating A Great Design Document"
  ---http://www.geocities.com/SiliconValley/Bay/2535/design_doc.html