

A Regular Language: The Annotated Case Report Form

Ryan Wilkins, PPD, Inc., Wilmington, NC

Joel Campbell, PPD, Inc., Wilmington, NC

ABSTRACT

As per CDISC and FDA regulatory requirements, including a link to the Annotated CRF (aCRF) from the metadata definitions facilitates a clear and transparent review of the clinical data supporting a regulatory submission [1]. In practice, providing this link can lead to a costly development of the electronic Case Report Tabulations (eCRT) or define.xml. This paper describes an automated approach rooted on the basis of Backus-Naur Form (BNF) to define the standard CDISC annotation construct. Then, using an intuitive design and a combination of tools – Adobe Acrobat, SAS XML Mapper, SAS Perl Regular Expression functions, and data step logic – programmatically applies and validates the links found in the define.xml at a drastically lower cost.

INTRODUCTION

Clinical Data Interchange Standards Consortium (CDISC) has driven the regulatory requirements to ensure that variable and value level annotations found in the Annotated Case Report Form (aCRF) are compliant both in form and syntax. Furthermore, guidance dictates corresponding hyperlinks from the metadata definitions are required when submission data originates from the CRF [2]. These requirements lead to the ability to review the metadata definitions or define.xml in such a manner as to efficiently trace data which originates from the CRF back to where it was captured. The best method to achieve this is to take the reviewer directly to the relevant page. In fact, as shown in the *Metadata Submission Guidelines, Appendix to the Study Data Tabulation Model Implementation Guide 3.1.2* [3], with minor modifications to the standard v1.0 define.xml style sheet, the technical hurdle of linking to a particular page is conquered. Therefore providing the link is not costly. The costly part resides in managing the annotation and data set metadata in such a way that:

1. Proper reference in the define.xml to the annotated variable is ensured.
2. Any variables which deserve annotation are included
3. Any variable annotations which are not part of the submission data are removed
4. Any variable or value level annotations which do not adhere to the CDISC guidance are corrected

Recent research and solutions (Rittman [4], 2010.; Yan [5], 2010.) show that many teams are managing define.xml metadata by way of Microsoft® Excel Spreadsheets or Workbooks. While this leads to a successful and natural approach to managing variable level metadata within a common document, it is not necessarily evident that it lends itself to the optimal approach to loading and verifying CRF Page references in the define.xml. Since the blank aCRF holds the annotated information and must be in a Portable Document Format (PDF -- blankcrf.pdf), it would be ideal to find a solution to export that information in such a way as to automatically load it into the define.xml. This automated approach would reduce the effort when data changes structurally and in the time taken to identify discrepancies between the two documents – the define.xml and aCRF.

The solution described in this paper derives itself from the basis of compiler construction which owes great dividends to John Backus and Peter Naur who introduced the formal notation known as Backus Naur Form (BNF) [6]. In reviewing the current guidance regarding the standard CDISC annotations, one can see how the annotations yield themselves to a formal language construct. One with which rules, expressions, and a regular grammar can define the set of standard annotations and thus, then be parsed to derive secondary data. This secondary data contains the variables, the value level annotations, and their corresponding pages. One can see how this data can then be used to achieve the four objectives identified above.

This paper is divided into 3 essential sections. After emphasizing the necessity to standardize the annotation construct, the first section, Importance of Standard Annotations, draws parallels on the relationship between the standard CDISC annotation set and how it relates to formal programming languages with the use of BNF or other, less formal notations. This section also shows, by way of examples, how to translate certain annotations into a notational format. The second section, The Design, describes the overall design and utility of this solution. The third section, Processing Order and Algorithms, drills down further, and details parsing the Adobe® Comment Export File (an XML file), the SAS® toolsets used and the complications or drawbacks which were met along the way. Each of

these equally important sections was written based on real-world, pilot projects and production use of the tool using Phase II and III clinical datasets and corresponding aCRFs. Again, the overall goal being to meet the four objectives identified above at a lower cost than existing methods.

IMPORTANCE OF STANDARD ANNOTATIONS

Incorporating the annotated CRF into the define.xml simplifies the metadata review and provides an inherently traceable transparent description of the data. Given that there must be some cost to provide this traceability and ease of review both in the time to build it and also, to validate it, automatic solutions should be leveraged to help lower this cost. Standardizing the annotation construct is key to approaching this programmatically.

THE CURRENT ANNOTATION STANDARD AND BNF

At an object level, the annotated CRF is a document with substantial fundamental data stored within. In this case, the annotation, the corresponding CRF and the CRF position within the PDF book is the secondary data available to extract and consume. On its face, the relationship of a simple annotation to that of a programming language or context free grammar may not be apparent. But at a closer look, a simple example shows that it is essentially that -- a language which can be described with notations and rules. In the context of SDTM data and their corresponding annotations, further rules and accordances are afforded. For example, other than a predefined set of exceptions, the first two characters of an annotation relate to that variable's parent domain e.g. "AE" in "AESTDTC". With the variable's parent domain, the primary key – domain and variable name – is now available to link the define.xml metadata to their corresponding annotated CRF locations.

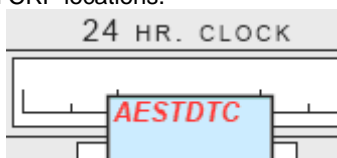


Exhibit 1. Single Variable Annotation

With this in mind, we need an annotation construct where, for a simple annotation, two things can be defined: (1) the domain and (2) variable. Notations to describe the annotated construct express the annotated language in a rule set form which acts as the foundation for developing an automated approach – a parser – that manages further extensions of the annotated grammar. Below is a simple single variable annotation defined in Backus-Naur Form; with this simple annotation defined, one can go further to define more complex annotation examples.

```
<variable1> ::= <dom> <term>
<dom> ::= <letter> <letter>
<term> ::= <letter> | <term> [ <letter> | <digit> ]
```

For instance, an annotation which corresponds to the value-level metadata related to Supplemental Qualifier data can be notated as:

```
<vlmsuppl> ::= "SUPP" | <dom> | ".QVAL where QNAM=" | <term>
```

Anytime the parser finds a term which matches this rule, it knows that the CRF page link in the define.xml metadata will correspond to a set of Supplemental Qualifier value level metadata (VLM). The key components to link the CRF page number of the annotation below in the define.xml metadata is (1) the domain, "SUPPAE"; (2) the variable, "QNAM"; and (3) the corresponding value, "CRELID".



Exhibit 2. Supplemental Qualifier VLM

Admittedly, the complexity of the more commonly used annotations does not venture much further than the one above. But, without a standard set of rules, examples like the above could be restructured, mean the same to the human eye, but be missed by the parser, if no rule matches it e.g. "SUPPAE.QNAM=CRELID". As one can see, the annotation has been shortened – the ".QVAL where" removed – but, it retains the same overall meaning. Nevertheless, a parser with an extended set of rules would catch it:

```
<vlmsuppl2> ::= "SUPP" | <dom> | ".QNAM=" | <term>
```

A RELATIONSHIP TO COMPILERS

Parsing annotations to derive a data set that maintains the CRF page numbers of the define.xml metadata may not meet a strict definition of a compiler yet, one can see a natural relationship. With a standard set of annotations, the parsing algorithms have the ability to rely on a rule set to consume the annotations, categorize, then extract and assemble the secondary data. In essence, what a compiler does at a high-level. Annotations draw a parallel with a lexical grammar where the lexical analyzer, in this case the aCRF parser, identifies lexemes or tokens that are then further used to make meaning out of an annotation. Although a diverse rule set is more flexible and captures more annotations than one with limited rules, an annotation could still slip through and not satisfy a single rule. For those cases, the parser, like a compiler with ill-formed syntax, should identify the ill-formed annotation so that the user can correct it.

In the introduction, four objectives were laid out as the cornerstones of what would be needed as far as requirements of this automated solution to aCRF metadata maintenance in the define.xml. The fourth, *Any variable or value level annotations which do not adhere to the CDISC guidance are corrected*, is managed through the parsing process. Since the ruleset is built from CDISC compliant annotations, any annotation which does not satisfy a single rule is identified by page number and escalated to the user to give way to a quick correction. The balance between defining a finite set of rules which constitute "compliant" annotations and "allowing" for extensions of the language may at times seem to be conflicting objectives. A first pass analysis of these two objectives tends to suggest that making a more diverse rule set or for "allowing" extensions of the language carries more positives than constraining the rules or set of compliant annotations. However, giving ground to more forms of grammar that carry the same meaning, adds complexity to any parsing algorithm. It is not evident that this complexity outweighs the cost of restraining the language. Backus-Naur Form notations clip complexity down at its knees so that simple grammar definitions act as the basis for specifications; specifications which are then used to develop program code to logically process the actual annotations.

MULTIPLE ANNOTATION FORMS

To further the point that a diverse rule set is needed to capture and extract many forms of annotations, examine a few different forms of annotations below. For each, situational or scenario based categorizations can be made:

1. Is the annotation of a single variable?
2. Is the annotation supposed to convey the value level metadata?
3. Does the annotation need to include conditional statements to accurately convey a particular meaning?
4. Could the annotation be ambiguous?

An example of value level metadata:

```
<vlmtestcd1> ::= <dom> | "ORRES where " | <dom> | "TESTCD = " | <term>
```




Exhibit 3. VLM Standard form

Another example of the same form:

```
<vlmtestcd2> ::= <dom> | "TESTCD = " | <term>
```




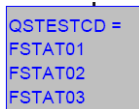
Exhibit 4. VLM Shorthand form

On occasion, the annotation may be ambiguous and for those, additional rules are required. For cases like VISIT or VISITNUM, the 2 character prefix which, in earlier examples, was assumed as the parent domain cannot be used under the same assumption here. An annotation of "VISIT" or "VISITNUM" is ambiguous; in these cases, the corresponding parent domain should precede the annotation and concatenated by a period e.g. PE.VISITNUM. Adding this prefix now gives the parser the ability to extract the domain which the annotation corresponds. In fact, for any ambiguous annotation, guidance supporting the annotated parser required ambiguous annotations to precede with a domain prefix and thus the rule for this example is as follows:

```
<variable2> ::= <dom> | "." | <term>
```

The challenges of defining a proper set of notational rules do not end with diversity in the grammar. Though to understand why, one must understand how the comment is stored in the blank aCRF (PDF). The next section will

cover the technical hurdle in more detail, but to think of it at a high level – each annotation or Adobe comment is stored as one continuous string. Annotations like the following work to quickly convey that, for the next 3 responses, each correspond to one value of QSTESTCD accordingly. But, the original notation (above) does not fully satisfy an annotation like the following. BNF could still describe an annotation of this form, but to simplify matters, the use of regular expressions along with BNF-like definitions seemed to work best.



```
QSTESTCD =
FSTAT01
FSTAT02
FSTAT03
```

Exhibit 5. VLM Recursive form

INTRODUCING REGULAR EXPRESSIONS

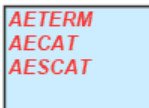
Regular expressions, in the context of computer science and compilers, provide a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters. For this purpose, it's a notational technique that simplifies the rule definition while still retaining sufficient information to accurately code this rule. The use of regular expression rules and key characters were implemented as follows:

- ? The question mark indicates there is *zero or one* of the preceding element.
- * The asterisk indicates there are zero or more of the preceding element.
- + The plus sign indicates that there is one or more of the preceding element.
- \$ Matches the ending position of the string or the position just before a string-ending newline.

To continue with the example in Exhibit 5, the notation can now be defined using the regular expression rules to show that the equals (“=”) after QSTESTCD can be followed by any *n* number of terms where *n* > 1 e.g.

```
<vlmtestcd2> ::= <dom> | "TESTCD = " | <term>+
```

Similarly, the following annotation can be defined as: `<variable1> ::= [<dom><term>]+`



```
AETERM
AECAT
AESCAT
```

Exhibit 6. Multi-Single Variable form

As one can see, incorporating this notational technique in the rule definitions shortens and gives flexibility especially when recursive annotations need to be described accurately and succinctly. Neither one of these notational schemes can be achieved without a standard set of compliant annotations and scenarios to which they apply. At a minimum, standardizing the CDISC annotation construct provides a common language which both the developer and regulatory reviewer understand. In actuality, defining the standard annotation construct in notational form leads way to programmatic solutions; solutions that use the annotations as language constructs that contain secondary data. Secondary data can be used to automatically load CRF page references into define.xml metadata. As such, the notations play a vital role in the development of such a system as they lay the foundation of this programmatic solution. The next section details the overall design of the self-validating system, the technical challenges faced in practice and how it relates as a subcomponent within the context of the overall define development tool.

THE DESIGN

In this section, a high-level view of the utility is described. To be clear, the solution described here plays part of a larger define development toolkit. This toolkit provides the user with full control over the submission data set metadata that the define.xml is built from. Even though this is a subcomponent of a larger utility, for the purposes of this paper, the approach described is considered stand alone. The descriptions in this section support that idea – that the design itself is something that could be imparted into other programmatic solutions.

The utility uses essentially two parameters as input objects – an extraction of the annotations to an XML Forms Data Formatted (XFDF) file and the source data which the annotation references. Adobe Acrobat Professional has a feature which allows the end user to extract the annotations by way of *exporting comments* to an XML-based file. The specific steps to export the comments are version dependent; in Acrobat v6.0, a user can export comments by selecting menu items: Document >> Export Comments >> Select .xdf as the file type >> Save

The xFDF file contains elements and nodes that hold attributes and descriptors about the annotations present in the blankcrf. These descriptors include a page number on which the annotation resides, a coordinate which corresponds to the physical location on the page, and the corresponding comment or annotation itself [7]. The SAS XML libname engine bridges the gap between the XML file in its current form and SAS; the XML map gives one the ability to create a dataset with an annotation, coordinate, and page reference from the xFDF. The coordinate is used to identify unique comment strings or annotations. The parsing algorithm uses this unique identifier to ensure that each annotation is kept whole thus retaining the original meaning.

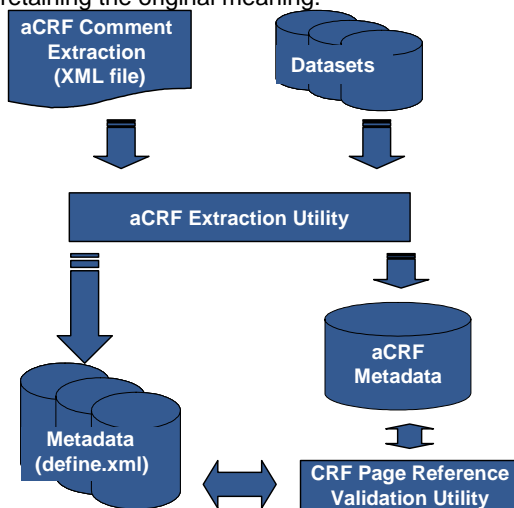


Exhibit 7. A High-level Process Flow

The aCRF Comment Extraction and source data are passed into the aCRF Extraction Utility. The rule sets described in the previous section have been implemented in SAS using a combination of regular expressions and other data step logic. More detail on this is found in the next section. Each comment is parsed and analyzed for meaning; then, if a meaningful and compliant annotation is found, the derived data is stored in a SAS data set. The SAS data set which holds this annotation metadata (“aCRF Metadata” in Exhibit 7) is structured such that each annotation is stored on a single record with variables and values that will provide a link to the define.xml metadata and the blank aCRF . This dataset and its members are described further in Table 1 below.

Dataset Name: ds_acrf_page	
Variable Names	Purpose
DOMAIN	The name of the dataset annotated e.g. DM or VS etc.
VARIABLE	The variable within the datasets e.g. RACE or VSTESTCD
VL_ITEM	The value related to a variable to which VLM corresponds in the define.xml. For instance, the value of QNAM, a Supplemental Qualifier variable or the value of any TESTCD variable - e.g. WEIGHT – taken from the annotation <i>VS.VSORRES where VSTESTCD='WEIGHT'</i>
PAGES	The variable contains a list of comma separated page numbers

Table 1. The contents of the annotated metadata stored as a data set

The aCRF metadata data set can be considered an operational data set. Not only is this data set used to merge on the CRF page references to the define.xml metadata, but it’s also used to validate these references once the define.xml is ready for production. As stated earlier, the end user has full control over the define.xml metadata by way of the define production tool. Since the user has full control, the user has the ability to mistakenly modify a CRF page reference in the metadata which, in the end, can cause the define.xml to link to the incorrect page or no page at all. If this is not caught before submitting the define for review, the quality of the submission diminishes. Also, if the aCRF is ever updated where more annotations are added, or removed, or changed, there needs to be a method to quickly include these changes and validate their correctness in the define. An operational dataset structured like the above contains the inherent ability to accomplish this task and the task of accomplishing the first three objectives outlined in the introduction. For the utility described in this paper, the operational data set – the annotated metadata – acts as the root component to successfully validate the define package as it relates to the blank aCRF . Further description on each of the validation objectives follows.

To accomplish the first objective, *ensuring all annotated variables are referenced properly in the define.xml*, the utility, immediately before building the define document, compares the define.xml metadata – specifically the variable origins – against the operational data set to ensure each page reference is correct. Any incorrectly referenced CRF pages

are identified so that the user can intervene and correct the page reference. For instance, if the define.xml metadata indicates “CRF Page 3” for data set “DM” and variable “BRTHDTC” yet, the annotated metadata has a value of “4” stored in the PAGES variable, the user is notified of this discrepancy.

In the utility, variables which originate from the CRF have a base value of “CRF” stored in their corresponding Origin define.xml metadata. This base or default value acts as the starting point to drive which variables should be annotated. If after running the aCRF parser against the annotated CRF extraction, variables may remain as “CRF”. Remaining “CRF” indicates that the variable may not have a corresponding annotation or may have an ill-formed annotation. In either case, the utility notifies the user of this discrepancy. As one can see, the 2nd objective – identify *variables which deserve annotation but are not included* in the blank aCRF – is accomplished.

Somewhat an inverse to the 2nd objective, the 3rd – *to identify any variable annotations which are not part of the submission data* – is accomplished through this same comparison. For instance, the aCRF extraction and parse could result in the following annotated metadata record. But, in this case, the variable, CMDECOD, is not included in the submission data. The comparison of annotated metadata against the define.xml metadata will then identify this orphaned annotation – an annotation not supported by the submission data. Before sending the define package, the user has a chance intervene and remove this annotation.

	pages	domain	variable	vl_item
24	8	CM	CMDECOD	

Exhibit 8. Snapshot of annotated data stored in the operational data set

The merit of accomplishing these three objectives stands on its own; that is, delivering a blank aCRF with correct and working CRF page hyperlinks in the define.xml is vital to the quality of submission material. Not surprisingly, after further study of CDISC guidance on managing the annotated CRF [8], one finds that these principles are not confined to this paper and the utility, but have already been defined as a requirement within guidance. A clear indication that CDISC and the regulatory review boards look for this correctness in the define and aCRF package.

PROCESSING ORDER AND ALGORITHMS

Now that the overall design has been described and the high-level path to automatically capture and validate CRF page references in the define has been laid forth, this section dives further into detail through describing the logical processing order of the annotations, the SAS functions used, and the technical challenges which were faced along the way. In the end, the final goal remains the same – to extract, parse, propagate and validate the annotated CRF page references in the define.xml at a lower cost than existing methods.

While a number of approaches may be efficient, the parsing algorithm employed here has a logical flow where it first categorizes the type of annotation then depending on this categorization, sends the annotation through a category-dependent set of algorithms and functions to parse and derive the secondary data related to the annotation. At the first step in the decision tree, the annotation is scanned for keywords which indicate that the annotation can be discarded. Keywords like, ‘NOT SUBMITTED’, ‘SEE ANNOTATIONS’, ‘SAME AS’ and ‘ON PAGE’ signify to the parser that the comment string can be discarded. After unusable annotations are discarded, the next stop along the decision tree is to determine whether the annotation corresponds to Value Level Metadata (VLM). If it does, then the comment or annotation is sent down a specialized processing path for VLM related annotations. If the annotation does not relate to VLM then the annotation is pushed through another set of rules to handle different forms of single variable annotations.

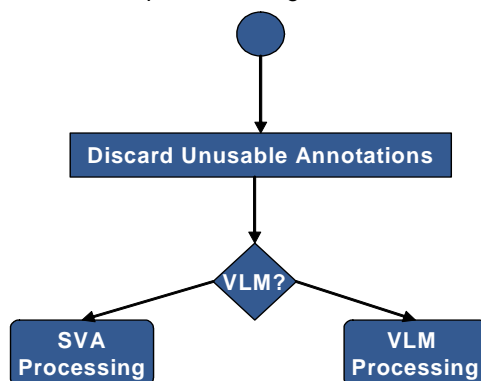


Exhibit 9. The initial processing flow of annotations

The single variable annotation and VLM processes then analyze the following set of rules to capture the domain, variable, and associated page number. In practice, the use of Perl Regular Expressions in SAS along with other SAS string functions proved to be a key component in these next steps. For additional discussion regarding the use of the SAS Perl regular expression functions in this application, see, *Importing and Parsing Comments from a PDF Document With Help From Perl Regular Expressions* (Campbell, 2011). By first categorizing the rule which the annotation satisfies, the parser can pass the comment to the proper lexical analyzer to derive the annotation into meaningful elements used to link and validate the CRF page references in the define.xml metadata.

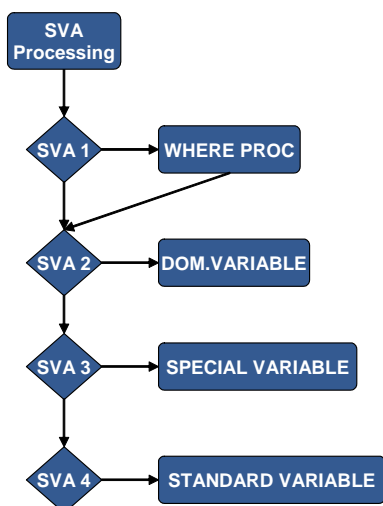


Exhibit 10. SVA Processing

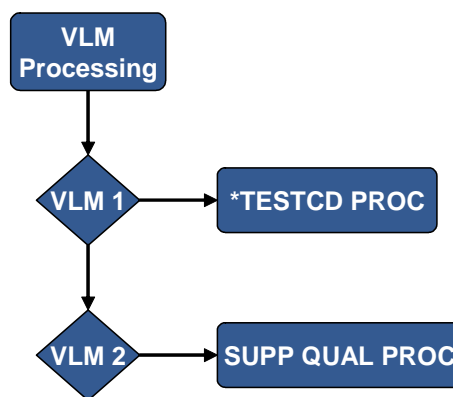


Exhibit 11. VLM Processing

Rule ID	Description
SVA 1	Identifies an annotation with a where clause embedded. Strips the variable out of the annotation for further processing
SVA 2	Identifies and parses any annotation that follows the form <DOMAIN>.<VARIABLE> where a period is used to concatenate the domain and variable together
SVA 3	Identifies a special set of variables which correspond to any domain – USUBJID, SITEID, etc. In the operational annotated metadata data set, an asterisk is used to indicate these variables correspond to any domain. Also, processing of Demographic variables – BRTHDTC, RFSTDTC, etc – which do not include the 2 character domain as the first two characters of the variable name.
SVA 4	Any standard SDTM variable where the first two characters in the variable name correspond to it's parent domain.
VLM 1	Identifies and parses any annotation that includes a *TESTCD variable followed by the values to which it equals.
VLM 2	Identifies and parses any annotation that corresponds to Supplemental Qualifier VLM

Table 2. Parsing rules and descriptions.

Even though defining the annotations in a BNF-like notation helped clear a development path to code the rules above, and the implementation of such was met with great success, the inevitable technical hurdles did arise. The following challenges were selected because, without solutions to each, the success of the utility would not be the same.

Along with the xFDF file format, Adobe can also provide an .FDF file that contains the PDF comments and corresponding page references. The .FDF document is an ASCII based text file. After careful exploration, it was determined that using the xFDF file format affords more flexibilities and ease of use with respect to parsing the extracted file and ensuring annotations are kept intact.

Handling ambiguous annotations was a particular problem early on in the development. Single variable annotations, for variables that could be found in multiple domains – like VISIT and VISITNUM – contained too many CRF page

references after the initial parsing algorithms completed. Annotation requirements were developed that mandated the use of the domain-dot prefix when annotating ambiguous variables e.g. QS.VISITNUM.

Because of their recursive nature, *short-hand* annotations like the one shown in Exhibit 5 initially proved challenging. The benefit to annotate in such a way – as it may save time – was the overarching argument to accommodate such annotations. Only after a methodical approach which used loops and regular expressions was the parser successful at extracting the relevant data for later use.

Early on in this paper a cost-reducing claim was made that this automated approach would lower the overall cost to develop and produce a define document. It is not the specific intent of this paper to compare current market estimates in relation to those found in practice while using the define production tool. Though on average, cost was reduced by a measure of 33% when compared to earlier methods to manage define.xml metadata and particularly, verifying the links within the define document. Even in a hypothetical sense, any cocktail napkin analysis comparing a manual validation technique – where the user verifies links by hand, selecting and confirming CRF page links in the define.xml -- versus an automated technique which does the same, over the long haul, should find that the automated approach has a lower cost. Furthermore, a manual technique carries the risk of human error and may not necessarily achieve all four objectives listed in the introduction.

CONCLUSION

At this point in the paper, one should have a stronger understanding of the overall architecture and technology used to automate a method to extract annotations from the blank aCRF, parse them for meaningful data, then assign and validate the CRF page references in the define.xml metadata. The methodology chosen for this approach draws parallels with compiler construction where defining the standard CDISC annotation construct in notational form provides a platform for growth and extensibility of the tool. It also promotes the growth of the annotation grammar to develop further standards.

Besides exhibiting the achievement of the four objectives that this automated approach must adhere to, this paper also shows how the utility has a positive impact both on the cost and quality that a manual approach cannot. In a clinical research marketplace, where cost and quality are of utmost concern, this approach allows for further extension of rules and guidance while retaining the four objectives.

This is not to say that every aspect to this problem – automated methods to managing the aCRF metadata -- has been solved. Areas like extending the annotation language construct to handle more scenarios where annotating the CRF adds value, reducing the amount of effort to annotate a CRF, or automatically correcting ill-formed annotations are worthy areas of study. Regardless of where this paper and overarching theme falls in the regulatory spectrum, our hope is that it has added value, if not at a theoretical level, but a practical one.

REFERENCES

- [1] CDISC Define.XML Guidance
<http://www.cdisc.org/define-xml>
- [2] FDA Study Data Specifications (SDS)
<http://www.fda.gov/downloads/Drugs/DevelopmentApprovalProcess/FormsSubmissionRequirements/ElectronicSubmissions/UCM163561.pdf>
- [3] Metadata Submission Guidelines, Appendix to the Study Data Tabulation Model Implementation Guide 3.1.2
<http://www.cdisc.org/stuff/contentmgr/files/0/4254da3fdc854804b27881ca82d24051/misc/metadata.zip>
- [4] *Automating the Link between Metadata and Analysis Datasets*, Rittmann, Risha.;
<http://www.lexjansen.com/pharmasug/2010/ad/ad16.pdf>
- [5] *Updating SDTM Metadata Excel File (Define.xls) with SAS*, Yan, Lin.;
<http://www.lexjansen.com/pharmasug/2010/cc/cc07.pdf>
- [6] *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*
<http://www.cs.uiowa.edu/~slonnegr/plf/Book/Chapter1.pdf>
- [7] *XML Forms Data Format Specification*
http://partners.adobe.com/public/developer/en/xml/xfdf_2.0.pdf
- [8] *CDISC Guidelines for Annotating CRFs*
<http://cdiscportal.digitalinfuzion.com>

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Ryan Wilkins
Enterprise: PPD Inc.
Address: 929 North Front Street
City, State ZIP: Wilmington, NC, 28405
Work Phone: +1 (910) 558 6013
Fax:
E-mail: Ryan.Wilkins@ppdi.com
Web:

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.