

Applying XML technology for implementation of natural language specifications

Beum-Seuk Lee Barrett R. Bryant

Department of Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham, AL 35294-1170, U. S. A.
{leebs, bryant}@cis.uab.edu

Abstract

By using the systematic structure of XML, the tangibility of the information of unrestricted natural language specifications is improved for the automation of the translation of these specifications into a formal representation. In the same context, specifying domain-specific knowledge explicitly in XML guides this automation regarding maintaining consistent vocabulary use and presenting implicit relations among components. Besides computational linguistics technologies applied in our research, XML representation is useful in identifying the ambiguity and errors in specifications. The result is a specification program that assists the user to specify, to prototype, and even to implement systems throughout the system development.

1 Introduction

Resources devoted to collect information and then to compile this information into a requirements document take a great portion of the entire system development life cycle. Both poor practices of the requirement composition and the informal and therefore ambiguous properties of the unrestricted requirement documents make it hard for the engineers to carry out correct interpretation of the specifications and the translation into a formal specification, not to mention employing the automation thereof.

It is well known that as much as 60 percent of the errors that appear during a system's life cycle have their origin in the requirements phase [6]. It is also well known that the cost to correct an error found in later stages of system development is orders of magnitude higher than to correct the same error found during the requirements stage [2]. Therefore ensuring the correctness of the requirements as well as their interpretation and translation cannot be overemphasized.

To deal with writing effective natural language requirements specification, Wilson pointed out the three areas where the major deficiencies were found after analyzing 40 approved NASA requirements specification documents; organization of requirements information, structure of requirements statements, and the language used to express requirements [17].

Here we propose an approach to resolve the informality and ambiguity problem in requirements specifications while avoiding the deficiencies mentioned above, by applying the systematic structure of XML and computational linguistics technology to automate the translation from requirements specifications written in natural language (NL) into a formal specification.

2 System Structure

The overall structure of our system is illustrated in this section by following each process step by step to show the role of XML. Figure 1 gives an overview of this process. To achieve the conversion from requirements

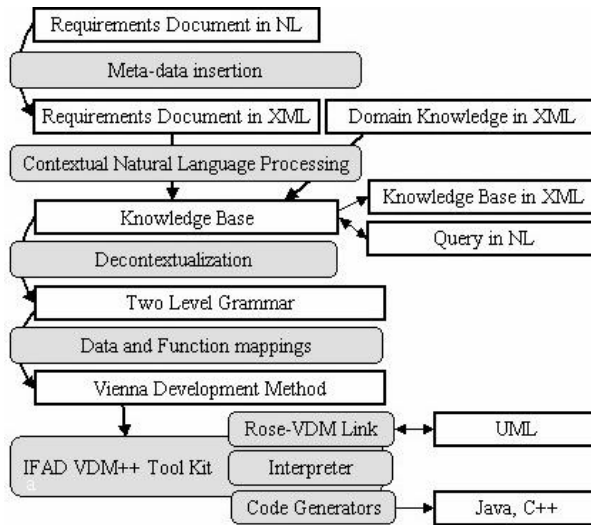


Figure 1: System Structure.

documents to a formal specification several levels of conversions are necessary. First the original requirements written in natural language is refined as a preprocessing of the actual conversion. This refinement task involves checking spelling, grammatical errors, consistent use of vocabularies, organizing the sentences into appropriate sections, etc. Next the refined requirements document is expressed in XML format. By using XML to specify the requirements, XML attributes (meta-data) can be added to the requirements to interpret the role of each group of sentences. XML is also used to specify the domain-specific knowledge describing the relationship among components and other constraints that are presumed in requirements documents or too implicit to be extracted directly from the original documents.

Then a knowledge base is built from the requirements document in XML using Natural Language Processing (NLP) [12] to parse the documentation and to store the syntax, semantics, and pragmatics information. In this phase, the ambiguity is detected and resolved, if possible. Once the knowledge base is constructed, its content can be queried in natural language. Next the knowledge base is converted, with the information of the domain specific knowledge in XML, into Two Level Grammar (TLG) [4] by removing the contextual dependency in the knowledge base. TLG, the most NL-like specification language which is a unification of functional, logic, and object-oriented programming styles, is used as an intermediate representation to build a bridge between the informal knowledge base and a formal representation.

Finally the TLG code is translated into VDM++ [9] (an object-oriented extension of the Vienna Development Method [1]) by data and function mappings. VDM++ is chosen as the target specification language because VDM++ has many similarities in structure to TLG and also has a good collection of tools for analysis and code generation. Once the VDM++ representation of the specification is acquired we can do prototyping of the specification using the VDM++ interpreter. Also we can convert this into a high level language such as JavaTM or C++ or into a model in the Unified Modeling Language (UML) [16] using the VDM++ Toolkit [11].

The translation of our system is incremental and iterative reflecting the changes made throughout the system development. User interaction is likely to happen any stage of the translation to supervise and assist the automation. By keeping track of the user's preferences and configurations for each iteration and by automating the translations accordingly, the user's repetitive involvement can be reasonably reduced.

In the sections which follow, we will present the main merits in utilizing XML to assist the system with natural language processing to achieve a smooth translation of an informal specification into a formal specification, keeping our focus mainly on the translation from natural language specifications to TLG.

3 With or without XML

In this section, we will attempt to advocate the use of XML in representing requirements specification data and its metadata by comparing two specifications, one with and the other without XML leverage, respectively. Even though our system can handle both cases, our recommendation is to use the one with XML as its rationale will become apparent in this section. An example of a simple Automatic Teller Machine (ATM) specification is used in this section for illustration purpose.

3.1 Without XML

The following is a natural language specification of an Automatic Teller Machine (ATM) in plain text.

```
Bank verifies ID and PIN giving the balance in the following order. It selects the
account from the list of accounts where the ID of the account is equal to the ID and
the PIN of the account is equal to the PIN. And then it assigns the balance of the
account into the balance.
```

```
It updates the balance with ID in the following sequence. It selects the account from
the list of accounts where the ID of the account is equal to the ID. And then it
assigns the balance to the balance of the account.
```

```
Automatic teller machine has 3 service types: withdraw, deposit, and balance check.
```

```
For each service first it verifies ID and PIN from the bank giving the balance.
```

```
ATM withdraws an amount with ID and PIN giving the balance in the following sequence.
```

```
If the amount is less than or equal to the balance then it decreases the balance by
the amount. And then it updates the balance in the bank with ID.
```

```
The machine deposits an amount with ID and PIN giving the balance in the following
order. It increases the balance by Amount and then updates the balance in the bank
with ID.
```

```
ATM checks the balance with ID and PIN giving the balance.
```

Usually it is the case that the unit or type information is easily neglected. Also the relations between components are mentioned implicitly, if not omitted. For example, the above specification implies that a bank keeps a list of accounts and that an account has three data fields: ID, PIN, and balance. However it is not easy to guess correctly the types of each field of an account just by reading the requirements document itself without consulting with the domain experts for the domain specific knowledge.

When processed by natural language processing, certain phrases and paragraphing play important roles to determine how the sentences are related to one another. For example, the phrase “in the following order” indicates that one or more of the following sentences will describe the detailed process with the order in mind. Also the phrase “For each service first” shows that before each service is carried out this action has to be taken first. The paragraphing or sectioning groups related sentences together, which is a major part of the preprocessing of the actual conversion. However there is obvious limitation and inevitable ambiguity in grouping sentences by using only paragraphing and particular phrases. For instance, the phrase ‘And then’ in the sentence “And then it updates the balance in the bank with ID” of the withdrawal service of ATM can be interpreted in two ways depending on how to relate the sentence to its previous sentence. It can mean that the balance has to be updated only when the current balance is sufficient. Alternatively it can be interpreted that the balance updating action is to be executed regardless the sufficiency of the balance. These types of the contextual information in the document should be detected and interpreted during the natural language processing.

3.2 With XML

The following is the same specification of ATM mentioned in the previous section except it is represented in XML with extra metadata which is to replace the contextual phrases.

```
<document>
  <section title = "Bank" meta = "object">
    <section title = "Verification">
      <sentence meta = "head">
        Bank verifies ID and PIN giving the balance.
      </sentence>
    <sentence>
```

```

    It selects the account from the list of accounts where the ID of the account is equal
    to the ID and the PIN of the account is equal to the PIN.
</sentence>
<sentence>
    And then it assigns the balance of the account into the balance.
</sentence>
</section>
<section title = "Balance update">
<sentence meta = "head">
    It updates the balance with ID.
</sentence>
<sentence>
    It selects the account from the list of accounts where the ID of the account is equal to the ID.
</sentence>
<sentence>
    And then it assigns the balance to the balance of the account.
</sentence>
</section>
</section>
<section title = "ATM" meta = "object">
<sentence meta = "pre_condition">
    Automatic teller machine verifies ID and PIN from the bank giving the balance.
</sentence>
<section title = "Withdrawal service">
<sentence meta = "head">
    ATM withdraws an amount with ID and PIN giving the balance.
</sentence>
<sentence>
    If the amount is less than or equal to the balance then it decreases the balance by the amount.
</sentence>
<sentence meta = "sub">
    And then it updates the balance in the bank with ID.
</sentence>
</section>
<section title = "Deposit Service">
<sentence meta = "head">
    The machine deposits an amount with ID and PIN giving the balance.
</sentence>
<sentence>
    It increases the balance by Amount and then updates the balance in the bank with ID.
</sentence>
</section>
<section title = "Balance checking">
<sentence meta = "head">
    ATM checks the balance with ID and PIN giving the balance.
</sentence>
</section>
</section>
</document>

```

Figure 2 displays a snapshot of this same XML representation requirements document viewer. The meta attribute in XML indicates the role of each paragraph. For example the attribute 'object' indicates that its element is the root of an object and its sub-trees are the specifications of the object. Other meta attributes replace the contextual phrases mentioned in the previous section. The attribute 'head' is used to mark the sentence as the rule head (which contains the function signature) and the following sentences under the same tree will be considered as its body statements. The attribute 'pre_condition' plays the same role as the contextual phrase "For each service first" in the previous section. It is a precondition for every service of 'ATM' because it is located under the ATM object root as shown in the figure. However if the sentence is placed (dragged-and-dropped) under a certain service (e.g. **Deposit Service**), it becomes a precondition just for that service. The attribute 'sub' replaces the contextual phrase "and then" in the previous section with a specific semantics that indicates that it is a sub-statement of the previous if statement avoiding ambiguity as previously mentioned. If this 'sub' attribute is used for a tree instead of a sentence, then all the statements of the tree will become the sub-statements of the previous if statement allowing flexibility in setting scopes.

We have specified these meta attributes for various types of functionality in requirements to cover a wide

Tree	Meta info
[-] Requirements	
[-] Bank	object
[-] Verification	
[-] Bank verifies ID and PIN giving the balance.	head
[-] It selects the account from the list of accounts where the ID of the account is equal to the ID ...	
[-] And then it assigns the balance of the account into the balance.	
[-] Balance update	
[-] It updates the balance with ID.	head
[-] It selects the account from the list of accounts where the ID of the account is equal to the ID.	
[-] And then it assigns the balance to the balance of the account.	
[-] ATM	object
[-] Automatic teller machine verifies ID and PIN from the bank giving the balance.	pre_condition
[-] Withdrawal service	
[-] ATM withdraws an amount with ID and PIN giving the balance.	head
[-] If the amount is less than or equal to the balance then it decreases the balance by the amount.	
[-] And then it updates the balance in the bank with ID.	sub
[-] Deposit Service	
[-] The machine deposits an amount with ID and PIN giving the balance.	head
[-] It increases the balance by Amount and then updates the balance in the bank with ID.	
[-] Balance checking	
[-] ATM checks the balance with ID and PIN giving the balance.	head

Figure 2: ATM with XML snapshot.

range of different requirements documents. The document type definition (DTD) for this specification in XML is shown as follows.

```
<!ELEMENT document (section)*>
<!ELEMENT section (section|sentence)*>
<!ELEMENT sentence #PCDATA>

<!ATTLIST section title CDATA #IMPLIED>
<!ATTLIST section meta CDATA #IMPLIED>
<!ATTLIST sentence meta CDATA #IMPLIED>
```

Using a tree-like structure in XML the specifications become more descriptive as the tree expands further. Organizing and representing the requirements document in XML according to the roles of the specifications of the system not only enhances understanding of specifications. In turn, this enforces well-organized requirements information and consistent requirements statements overcoming the deficiencies in organization of requirements information, structure of requirements statements, and the language used to express requirements.

4 Domain-specific knowledge in XML

A requirements document usually contains specific information about how the system should work whereas the domain knowledge describes how the system is composed by its components and the constraints imposed on the components or on the relations among them. Therefore it is very common that a requirements document lacks explicit mentioning of the domain-specific knowledge or assumes that the reader is familiar with or has an easy access to this knowledge. Domain-specific knowledge is world knowledge specific to a certain domain in which the system is defined. This is well tied into the concept of the families or the ontology of systems. Depending on the level of abstraction (or the details described) of the domain knowledge, the effort to construct it may vary. By lifting up the level of abstraction, the body of the knowledge can be reduced into a reasonable size and so can the effort to build it. Usually the domain-specific knowledge is defined informally or only for a specific project, not reusable or extensible for similar systems (the systems in the same family). By using XML to specify the domain knowledge with a minimum semantics, not only

can the specification be formally defined but also it can be extensible, gradually building up an ontology of systems.

In our research the domain knowledge specified in XML shares many similarities with DARPA Agent Markup Language (DAML) [7] which is a frame-based language with semantics to describe ontology. Because domain knowledge is more than just an ontology, DAML is not expressive enough to describe all aspects of the domain knowledge [14]. However using the XML syntax a domain knowledge can be specified in various ways leaving the interpretation of its semantics entirely up to the system that uses it [5]. Therefore when a specification for domain-specific knowledge in XML is to be developed, its formal semantics as well as its expressiveness has to be considered at the same time.

The following describes an example of the domain knowledge of Car to illustrate the use of domain-specific knowledge expressed in XML in our project.

```
<system name = "Car">
  <composition name = "Engine" >
    <amount type ="exactly" value = "1"/>
    <unit type = "volume" value = "liter"/>
    <composition name="Cylinder" type = "integer">
      <amount type ="one_of" value = "4,6,8"/>
    </composition >
    <association with = "Starter" type ="pass_to" value ="signal"/>
  </composition >
  <composition name = "Wheel"/>
  <composition name = "Body">
    <onoma name = "Frame" type ="synonym"/>
  </composition >
  <inheritance type ="parent" of = "Van" />
  <inheritance type ="child" of = "Vehicle" />
</system>
```

According to the above domain specification, Car is composed of Engine, Wheel, Control, and Body. Vehicle is a parent of Car whereas Van is one type of Car. Car can have exactly one Engine and the unit of Engine is volume expressed in liters. Engine has Cylinder as its subcomponent. The number of Cylinders, which is as an integer number and is a representative part of this subcomponent, can be either 4, 6, or 8. Starter passes a signal to Engine (to turn on the motor). Body of Car also can be called Frame. Here the element 'onoma' which literally means name in Greek, conveys the information such as acronyms, hypernyms, and synonyms.

The following is a DTD for the domain knowledge in XML, which defines the formal semantics of the domain-specific knowledge while pertaining proper expressive power.

```
<!ELEMENT system (composition|association|inheritance|onoma)*>
<!ELEMENT composition (amount?, unit?, (composition|association|inheritance|onoma)*)>
<!ELEMENT association EMPTY >
<!ELEMENT inheritance EMPTY >
<!ELEMENT onoma EMPTY >
<!ELEMENT amount EMPTY>
<!ELEMENT unit EMPTY>

<!ATTLIST system name CDATA #REQUIRED>
<!ATTLIST composition name CDATA #REQUIRED type CDATA #IMPLIED>
<!ATTLIST association with CDATA #REQUIRED type CDATA #REQUIRED value CDATA #IMPLIED>
<!ATTLIST inheritance of CDATA #REQUIRED type (parent|child) #REQUIRED>
<!ATTLIST onoma with CDATA #REQUIRED type (acronym|hypernym|synonym) #REQUIRED >
<!ATTLIST amount type CDATA "exactly" value CDATA #REQUIRED>
<!ATTLIST unit name CDATA #IMPLIED type CDATA #REQUIRED>
```

Note that the domain-specific knowledge in XML for the translation doesn't have to describe the domain exhaustively. Namely most of the elements and attributes are optional and attribute values can be any character string. For example, the association element can represent message passing, priority, arbitrary relation between components, etc. The minimum information required to guide the translation would be sufficient with the possibility of adding on more information later when necessary.

The domain knowledge for our ATM example is shown as follows.

```

<system name = "ATM system">
  <composition name = "Bank" type = "component">
    <composition name = "AccountsList" type = "accountlist"/>
  </composition>
  <composition name = "Account" type = "component">
    <composition name = "ID" type = "integer"/>
    <composition name = "PIN" type = "integer"/>
    <composition name = "Balance" type = "real"/>
  </composition>
  <composition name = "ATM" type = "component">
    <onoma name = "Machine" type = "hypernym"/>
    <onoma name = "Automatic teller machine" type = "acronym"/>
  </composition>
</system>

```

The above specification describes that the whole system is composed of by `Bank`, `Account`, and `Automatic teller machine`. `Bank` keeps an `Accounts list` and each `Account` has three data fields : `ID`, `PIN`, and `Balance`. The type of each data field is also specified. `Automatic teller machine` is one type of `Machine` that can be abbreviated as `ATM`.

In the natural language documents one concept may be represented in many different ways causing the translation difficulty in clustering similar information together. This is closely related to the last deficiency mentioned in the introduction section, namely the language used to express requirements. There can be acronyms, synonyms, and hypernyms. From the ATM example, the word ‘ATM’ is interchangeable with ‘machine’ or ‘Automatic teller machine’. By using a minimum set of representative words that describes the entire components in the domain-specific knowledge, one-to-many relations between words and their various representations can be obtained and thus provides a simpler source to translate. The full set of words in the requirements documents are mapped into the minimum set of representative words by measuring similarity among words. The hypernym and the location of the common words are used for this estimation.

In summary, by specifying domain-specific knowledge in XML and limiting the scope of the knowledge the effort needed to build up the domain knowledge for the translation can be greatly reduced.

5 Conversion from XML to knowledge base

The raw information of the requirements document in natural language is not in the proper form to be used directly because of the ambiguity and implicit semantics in the document. Therefore an explicit and declarative representation (knowledge base) is needed to represent, maintain, and manipulate knowledge about a system domain [13]. Not only does the knowledge base have to be expressive enough to capture all the critical information but also it has to be precise enough to clarify the meaning of each knowledge entity (sentence). In addition, the knowledge base has to reflect the structure of TLG into which the knowledge base is translated later.

The knowledge base isn’t a simple list of sentences in the requirements document. The linguistic information of each sentence such as lexical, syntactic, semantic, and most importantly discourse level information has to be stored with proper systematic structure.

Each sentence of the requirements documents has to be represented in a way that eases the interpretation of the sentence. In computational linguistics, this is carried out by constructing a parse tree of the sentence, that contains the syntactic information of the sentence. By using this semantic information we can tell what type of operation (the predicate) a certain object (the subject) executes on other objects (the objects).

To build a parse tree, each sentence in the requirements document is read by the system and tokenized into words. At this point, the acronyms, hypernyms, and synonyms defined in the XML domain-specific knowledge are used to make words mappings. At the syntactical level, the part of speech (e.g. noun, verb, adjective) and the part of sentence (e.g. subject and object) of each word are determined by standard parsing techniques [12]. The corpora of statistically ordered parts of speech (frequently used ones being listed first) of about 85,000 words from Moby Part-of-Speech II [10] are used to resolve the syntactic ambiguity when there is more than one valid parsing tree. Our system is able to handle elliptical compound phrases, comparative phrases, compound nouns, and relative phrases to allow the natural language in the requirements documents to be as less controlled thus more natural as possible.

Tree	Meta i...
[-] Knowledge base	
[-] bank	context
[-] {account}{list}{keep}	bank
[-] bank keeps list of accounts	
[-] {balance}{order}{ID}{PIN}{verify}	bank
[-] bank verifies ID and PIN giving balance in following order	
[-] bank selects account from list of accounts where ID of account is equal to ID and ...	
[-] then bank assigns balance of account into balance	
[-] {balance}{ID}{sequence}{update}	bank
[-] bank updates balance with ID in following sequence	
[-] bank selects account from list of accounts where ID of account is equal to ID	
[-] then bank assigns balance to balance of account	
[-] account	context
[-] {three}{data}{field}{ID}{PIN}{balance}	account
[-] account has three data fields : ID, PIN, balance	
[-] ID and PIN are integers and balance is real number	
[-] ATM	context
[-] {service}{type}{withdraw}{deposit}{check}	ATM
[-] ATM has 3 service types : withdraw, deposit, balance check	
[-] {balance}{ID}{PIN}{bank}{service}{verify}	ATM
[-] first ATM verifies ID and PIN from bank for each service giving balance	
[-] {balance}{sequence}{amount}{ID}{PIN}{withdraw}	ATM
[-] ATM withdraws amount with ID and PIN giving balance in following sequence	
[-] if amount is less than or equal to balance then ATM decreases balance by amount	
[-] then ATM updates balance in bank with ID	
[-] {balance}{order}{amount}{ID}{PIN}{deposit}	ATM
[-] ATM deposits amount with ID and PIN giving balance in following order	
[-] ATM increases balance by Amount and then ATM updates balance in bank with ID	
[-] {balance}{ID}{PIN}{check}	ATM
[-] ATM checks balance with ID and PIN giving balance	

Figure 3: Knowledge base for ATM without XML.

Also the anaphoric references (pronouns) in a sentence are identified according to the current context history. A pronoun can represent a word, sentence, or even context. It is worthwhile to mention here that the requirements documents are easier to process than other types of textual documents in the sense that usually requirements documents have well defined structures with less ambiguities and infrequent use or narrow reference scope of pronouns.

Once the references of pronouns are determined, each sentence is stored into the proper context in the knowledge base. The structure of the knowledge base reflects the structure of the requirements in XML. The meta attribute information from XML is also stored in the knowledge base to be used for the translation from the knowledge base into TLG. As for the case of the requirements without XML representation, the system totally relies on the linguistic information in the document to build the knowledge base according to the context. For more information on this process, we refer the readers to [15].

The knowledge base which is constructed from the ATM requirements without XML representation is shown in Figure 3.

In this figure some intermediate information is shown, which is used to build up the knowledge base structure with the aid of contextual phrases and paragraphing. The linguistic information is not shown in the figure except the Ids of the sentences in the database that hold this information and the sentences that are generated from the information in the database.

In summary, the knowledge base stores not only the linguistic information of each sentence but also the

data structure and meta information of related sentences as specified in the requirements in XML. Along with this process, linguistic ambiguity is detected and resolved in parsing and construction of the knowledge base.

6 Transition from knowledge base to TLG

Two-Level Grammar (TLG) may be used to achieve translation from an informal natural language specification into a formal specification. Even though TLG has NL-like syntax, its notation is formal enough to allow formal specifications to be constructed using the notation. It is able not only to capture the abstraction of the requirements but also to preserve the detailed information for implementation. The term “two level” comes from the fact that a set of domains may be defined using context-free grammar, which may then be used as arguments in predicate functions defined using another grammar. TLG may be used to model any type of software specification. The basic functional/logic programming model of TLG is extended to include object-oriented programming features suitable for modern software specification [3]. The syntax of the object-oriented TLG is:

```
class Class_Name.  
  Data_Name {, Data_Name}::Data_Type {, Data_Type}.  
  Rule_Name : Rule_Body {, Rule_Body}.  
end class [Class_Name].
```

where the term that is enclosed in the curly brackets is optional and can be repeated many times, as in Extended Backus-Naur Form (EBNF). The data types of TLG are fairly standard, including both scalar and structured types, as well as types defined by other class definitions. The rules are expressed in natural language with the data types used as variables. The conversion from the knowledge base to TLG flows very nicely because the knowledge base is built with the structure taking this translation into consideration. The root of each context tree becomes a class. And then the body of each class is built up with the sentence information in the sub-contexts of the root. Combined with the specification in the domain-specific knowledge, the knowledge base of the ATM example would be translated into the following TLG specification.

```
class Bank.  
  
  AccountsList :: AccountList.  
  Balance :: Float.  
  ID :: Integer.  
  PIN :: Integer.  
  
  -- [ Verification ]  
  -- Bank verifies ID and PIN giving the balance  
  verify ID and PIN giving Balance :  
    -- It selects the account from the list of accounts where the ID of the account is equal to the ID  
    -- and the PIN of the account is equal to the PIN  
    select Account from AccountsList with ID of Account = ID and PIN of Account = PIN,  
    -- And then it assigns the balance of the account into the balance  
    Balance := Balance of Account.  
  
  -- [ Balance update ]  
  -- It updates the balance with ID  
  update Balance with ID :  
    -- It selects the account from the list of accounts where the ID of the account is equal to the ID  
    select Account from AccountsList with ID of Account = ID,  
    -- And then it assigns the balance to the balance of the account  
    Balance of Account := Balance.  
  
end class.  
  
class ATM.  
  
  Balance :: Float.  
  Amount :: Float.  
  ID :: Integer.  
  PIN :: Integer.
```

```

-- [ Withdrawal service ]
-- ATM withdraws an amount with ID and PIN giving the balance
withdraw Amount with ID and PIN giving Balance :
  -- Automatic teller machine verifies ID and PIN from the bank giving the balance
  verify ID and PIN from Bank giving Balance,
  -- If the amount is less than or equal to the balance then it decreases the balance by the amount
  if Amount <= Balance then
    Balance := ( Balance - Amount ),
    -- And then it updates the balance in the bank with ID
    update Balance in Bank with ID
  endif.

-- [ Deposit Service ]
-- The machine deposits an amount with ID and PIN giving the balance
deposit Amount with ID and PIN giving Balance :
  -- Automatic teller machine verifies ID and PIN from the bank giving the balance
  verify ID and PIN from Bank giving Balance,
  -- It increases the balance by Amount and then updates the balance in the bank with ID
  Balance := ( Balance + Amount ),
  update Balance in Bank with ID.

-- [ Balance checking ]
-- ATM checks the balance with ID and PIN giving the balance
check balance with ID and PIN giving Balance :
  -- Automatic teller machine verifies ID and PIN from the bank giving the balance
  verify ID and PIN from Bank giving Balance.

end class.

class Account.

  ID :: Integer.
  PIN :: Integer.
  Balance :: Float.

end class.

```

When the system proceeds with the ATM knowledge base, it detects the fact that the data type of the Amount hasn't been explicitly specified and suggests 'Float' as its first choice among other types after spotting its use in arithmetic with Balance whose type is declared as Float in the domain-specific knowledge in XML. In addition to the type information, the relations among components are extracted from the domain-specific knowledge and reflected in the final TLG code; a list of accounts is a component of a bank and ID, PIN, and balance are data fields for each account. Also observe that the sentence that increases or decreases the balance is mapped into the TLG assign statement. Natural language has a fairly large size vocabulary whereas TLG uses specific words for the language-defined operations. Therefore there is a many-to-one mapping, which the user has to specify, from a natural language expression to a specific TLG operation just like the example of the increase and decrease operations. The rules for this mapping are highly parameterized for flexibility. For example, the following rule states that if the verb of a sentence is 'assign' and its first preposition is either 'to' or 'into', translate this into a TLG statement where the first object of this preposition is assigned to the first object.

```
verb:assign,obj1:#A,prep1:to|into,prepobj1:#B => #B := #A
```

By applying the above rule, the sentences "it assigns the balance of the account into the balance." and "it assigns the balance to the balance of the account." are mapped to TLG statements "Balance := Balance of Account" and "Balance of Account := Balance", respectively.

After the conversion from the knowledge base into TLG, the TLG code is translated into VDM++ by data and function mappings (for more details on this translation we refer the readers to [4]). Once we have translated the TLG specification into a VDM++ specification we can convert this into a high level language such as JavaTM or C++, using the code generator that the VDM++ ToolkitTM provides. Not only is this code quite efficient, but it may be executed, thereby allowing a proxy execution of the requirements. This

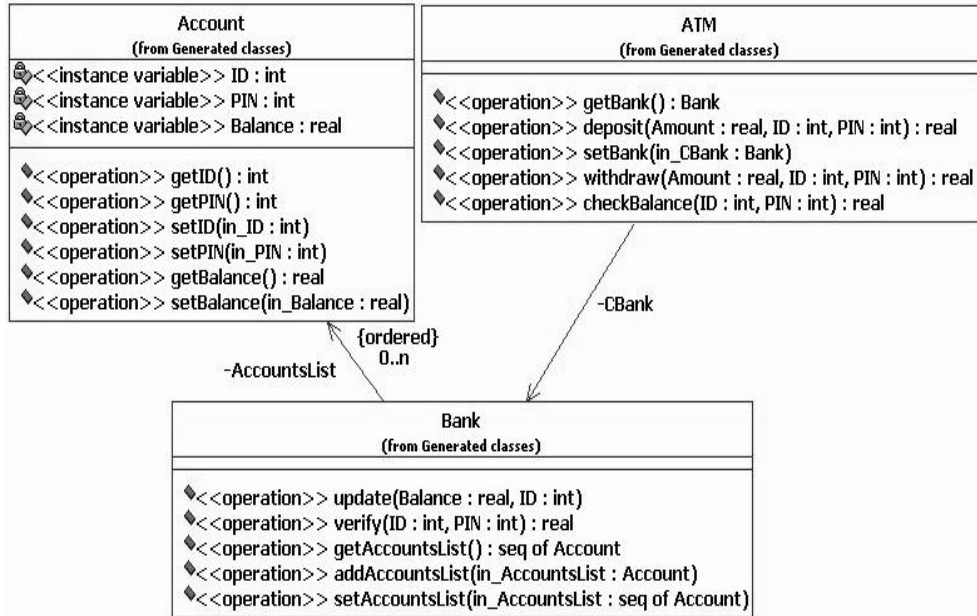


Figure 4: UML for ATM.

allows for a rapid prototyping of the original requirements so that these may be refined further in future iterations. Namely the inconsistencies, contradictions, and ambiguities hidden in the informal description can be discovered in the formal representation using the VDM++ Toolkit. Another advantage of this approach is that the VDM++ Toolkit also provides for a translation into a model in the Unified Modeling Language (UML) using a link with Rational RoseTM (Figure 4).

7 HbsAG, a real-world specification

In this section, we will show the result we have obtained after processing a natural language requirements document of Hepatitis B Surface Antigen (HbsAg) management [8] and confirm how our system assists the user throughout the life cycle of system development.

7.1 HbsAG, how well defined?

At first glance, the HbsAg requirements document (see the appendix A) seems very logical and easy to follow with constancy in both the terms and the phrases used. The structure of the document itself looks convincingly well laid out with proper levels of numbering. Also it is worth noting that there is a very well defined separation between specifications and the commentary notes. Therefore on the surface the document seems written effectively.

After processing the document, our system detected the following deficiencies in the document, which can generate ambiguities (an exception to this is a minor typographical error).

A typographical error such as the word ‘than’ instead of ‘then’ in the sentence of the section III.b can be easily fixed and won’t cause any serious misinterpretation. However there may be some cases where a misspelling can lead to a hazard. Even though this kind of error is supposedly fixed at the preprocessing before any conversion takes place, some of them can be detected during natural language processing. An unpaired parenthesis or bracket can also mislead the reader. For example, the footnote 3 appended to the section III.c doesn’t have its closing bracket, making it hard to decide which statements to include as a footnote statements. We took all the rest of the statements as a part of the footnote as it is the case for all other footnotes throughout the document. The most serious problem we found in the document was the confusing use of the terms. For instance, the terms “test result”, “donor”, “donation”, and “sample” are

used as if they were interchangeable (this is different than the case of the phrases “non-reactive” and “not reactive” both of which have the same semantics). Only one of these terms has to be selected when it is to be passed as a parameter for a function or assigned as a variable for a class. As a matter of fact, this is one of the most common but significant problems when unrestricted textual documents undergo an automation. As mentioned previously, in formal specification, terms or variables have to be used consistently whereas in most of the natural language requirements documents this is rarely practiced, mixing terms at will. For now, to resolve this problem, our system relies on the preprocessing, by maintaining mapping rules between terms, and hypernyms, acronyms, synonyms in the domain-specific knowledge in XML. Also the relations among components specified in the domain-specific knowledge help the readers to interpret the document correctly when encountering ambiguous terms. Construction of an ontology for software system vocabulary to maintain consistent terminologies is under consideration for our future work. This HbsAg requirements document refers to some of the previously stated procedures. This was possible because the document is well organized with proper sectioning and numbering. However this also can cause an ambiguity. For example, in the introduction part of the section IV, the document states that “if reactive, retested in duplicate for HbsAg and for anti-Hbc as indicated in Sections I and III, respectively, above” referring back to the previous sections. However from this statement it can be easily misinterpreted that every single statement in those stated sections are meant to be carried out (for example, acceptance of the donor when a condition is met) when, in reality, only the statements that update the status of the sample (whether or not the sample is reactive for a specific test) are intended for reuse. In addition, there is an ambiguity when Section I is reused, which has a control path to the section II. Because the section II is not included in the referenced sections (that is section I and III), it is not clear if we have to ignore this case. This also happens frequently in requirements documents, when the domain experts try to avoid repetitive statements and to reuse the previously mentioned statements, resulting in looping infinitely or execution of unintended statements. It is worthwhile to note the fact that there are many errors that are caused by the ambiguities in the requirements document in addition to the logical errors within the system control flows. However in general the latter is more emphasized than the former as the case for efforts put into verification.

In summary, many deficiencies are easily overlooked when the requirements are in an informal format such as an unrestricted textual natural language. However these problems are detected and resolved, when possible, throughout the automatic conversions by our system to formally state the specifications. This can be understood by looking at the problem from a different angle. Namely, the deficiencies in the requirements document may merge because the person who writes the document doesn't have its automated conversion to a formal specification in mind and lacks instruction and guidance for carrying that out.

7.2 HbsAG, from informal to formal

As shown in Appendix B, the knowledge base of the HbsAg requirements closely resembles the original textual requirements as the original document itself is well structured (its corresponding snapshot can be found in Appendix C). The preprocessing has rephrased some sentences to keep phrasings and terms consistent and simple, after fixing minor misspelling. For example, the sentence “see Section V” is substituted by the sentence “check for inconclusive case” and the phrase “sample drawn at this time” is replaced by the word “subsequent sample.” And as you may have noticed, there are a couple of new meta attributes. The meta attribute ‘otherwise’ turns the current sentence (or a block of statements) into an else-statement pairing it with its previous if-statement. This is another beauty of using the XML syntax as else-statements and their scopes are difficult to specify without ambiguity in natural language sentences (and sometimes even in programming languages, e.g. the dangling else problem in C++, Java, etc.). The meta attribute ‘comment’ comments out the specified sentence. The original natural language specification is inserted as a comment for each TLG statement. If a sentence is marked as a comment in the XML requirements document, this comment will be inserted in the TLG code without its corresponding TLG statement just for future reference.

The final TLG code may be found in Appendix D. For this particular document, it is almost impossible to define relationships between different components (sample, donor, unit, etc.) without altering the basic structure of the requirements. So only the types of the components are loosely defined. Having poorly defined relations between components is another place that needs a principal improvement in requirements composition.

There are defined three TLG mapping rules to convert natural language sentences into TLG code.

```

subj:Sample,verb:be,comp:#A,not:#B=not => Sample = "#B #A"
subj:Sample|Subsequent_Sample,subj:#A,verb:be,comp:#B,prep1:for,prepobj1:#C,not:#D=not => #A = "#D #B for #C"
verb:treat,obj1:Unit,prepl:as,prepobj1:#A,prep2:for,prepobj2:#B => Sample := "#A for #B"

```

The first rule says that any sentence whose subject is ‘Sample’ and the verb is a ‘be’ verb with a complement (the value of this complement is assigned to the variable #A), is to be mapped into TLG code in which the complement equals to Sample and if there is a negative adverb, the word ‘not’ is put right after the opening quotation symbol (by assigning the word ‘not’ to the variable #B). By applying this rule, for example, the sentence “the sample is not neutralizable” would be mapped to the TLG statement `Sample = “not neutralizable”`.

The second rule is almost identical to the first rule except that the subject can be either ‘Sample’ or ‘Subsequent_Sample’ and the object of the first preposition is included in the TLG code if the preposition is ‘for’. After applying this rule to the sentence, for example, “ the subsequent sample is reactive for anti-HBc” we would get its corresponding TLG statement `Subsequent_Sample = “reactive for anti-HBc”`.

The last rule replaces any sentence that looks like “treat Unit as something” with a TLG assignment statement `Sample := “something”`. So the sentence “treat the unit as reactive for Anti-HBc” would have its corresponding TLG statement `Sample := “reactive for Anti-HBc”`.

The final XML specifications and its TLG code of our HBsAg requirements document after defining the relations among the components may be found in Appendix E (its snapshot is in Appendix F) and Appendix G respectively. The domain-specific knowledge in XML of this specification is as follows:

```

<system name = "HBsAg system">
  <composition name = "Donor" type = "component">
  <composition name = "Unit" type = "undefined"/>
  <composition name = "Sample" type = "string"/>
  <composition name = "Subsequent_Sample" type = "string"/>
</composition>
</system>

```

The main procedure that carries out appropriate actions according to the result of each test, is separated from each specific test to avoid the ambiguous section referencing. Also this allows some of the conditional statements with the same consequences to be merged together, making the specification simple. For the sake of programming language semantics, input and output variables are explicitly stated. For example, the rule “test Sample for HBsAg returning Sample” takes ‘Sample’ as its input and returns a value and assigns it to the variable ‘Sample’. Before this rearrangement, every variable was globally accessible. Now every variable is defined and accessible only locally in each rule of the class Tests as the result of having the relations among components and separating the rules that updates control variables from the main rule that acts upon those variables. The UML representation of the final HBsAg is shown in Figure 5.

In summary, using our system, the linguistic ambiguity and programming language logic errors that are too evasive to be spotted in the natural language requirements are detected for further improvement and implementation. The result from the real world specifications of HBsAg sets forth a set of recommendable practices in composing requirements. Also having a specific target representation (for automation, for example) in mind helps to compose well-structured and consistent specifications.

8 Contribution and Conclusion

This research project is developed as an application of formal specification and computational linguistic techniques to automate the conversion from a requirements document written in natural language to a formal specification language while assisting the developers with repetitive tasks.

XML is used to define the contextual information and to organize the specification with well-defined structure to supplement the shortcomings of the original unrestricted natural language specifications. Also formally defined XML syntax is employed to express the implicit domain-specific knowledge with flexibility and expressiveness. Due to its NL-like flexible syntax without losing its formalism, TLG is chosen as a formal specification to fill the gap between the different level of formalisms of NL and formal specification languages.

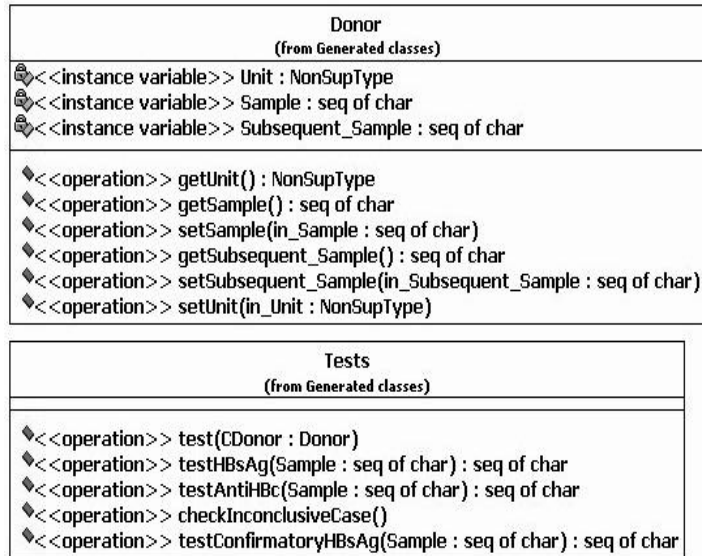


Figure 5: UML for HBsAg.

The system provides a very useful tool to assist software engineers in moving from the requirements document to the formal specification. Our future work is to continue developing the system to improve system usability and robustness with respect to its coverage of requirements documents. When finalized, it is expected that by using the formalized context in NLP and TLG as a bridge between the requirements document and a formal specification language, we can achieve an executable and reusable NL specification for a rapid prototyping of requirements, as well as development of a final implementation assisting the developers throughout the software development life cycle.

Acknowledgements. This material is based upon work supported by, or in part by, the U.S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number DAAD19-00-1-0350 and by the U. S. Office of Naval Research under award number N00014-01-1-0746. The authors would like to thank IFAD for providing an academic license to the IFAD VDM Toolbox in order to conduct this research.

References

- [1] D. Bjørner and C. B. Jones. *The Vienna Development Method: The Meta-Language*. Springer-Verlag, 1978.
- [2] B. W. Boehm. Software Engineering Economics. *IEEE Transactions on Software Engineering*, 10(1):4–21, January 1984.
- [3] B. R. Bryant. Object-Oriented Natural Language Requirements Specification. *Proc. ACSC 2000, 23rd Australasian Comp. Sci. Conf.*, pages 24–30, 2000.
- [4] B. R. Bryant and B.-S. Lee. Two-Level Grammar as an Object-Oriented Requirements Specification Language. *Proc. 35th Hawaii Int. Conf. System Sciences*, Jan. 2002.
- [5] J. C. Cleaveland. *Program Generators with XML and Java*. Prentice-Hall, 2001.
- [6] A. Davis. *Software Requirements Analysis and Specification*. Prentice-Hall, 1990.
- [7] S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74, 2000.
- [8] Director of Office of Biologics Research and Review, Food and Drug Administration. Recommendations for the Management of Donor and Units. <http://www.fda.gov/cber/blmem/120287.pdf>, 1987.
- [9] E. H. Dürr and J. van Katwijk. VDM++ - A Formal Specification Language for Object-Oriented Designs. *Proc. TOOLS USA '92, 1992 Technology of Object-Oriented Languages and Systems USA Conf.*, pages 63–278, 1992.
- [10] W. Grady. Moby Part-of-Speech II (data file), 1994.
- [11] IFAD. The VDM++ Toolbox User Manual. Technical report, IFAD (www.ifad.dk), 2000.
- [12] D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

- [13] G. Lakemeyer and B. Nebel. *Foundations of knowledge representation and reasoning*, volume 810. Springer-Verlag Inc., 1994.
- [14] B.-S. Lee and B. R. Bryant. Contextual Natural Language Processing and DAML for Understanding Software Requirements Specifications. *Proc. 19th International Conference on Computational Linguistics*, pages 516–522, 2002.
- [15] B.-S. Lee and B. R. Bryant. Contextual Knowledge Representation for Requirements Documents in Natural Language. *Proc. FLAIRS 2002, the 15th International Florida AI Research Symposium*, pages 370–374, 2002.
- [16] T. Quatrani. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley, 2000.
- [17] W. M. Wilson. Writing Effective Natural Language Requirements Specifications. Technical report, Naval Research Laboratory, 1999.

Appendix A - Original HBsAg specification in natural language

These recommendations set forth a series of procedures by which an initially HBsAg reactive donor may be reevaluated by a blood establishment, providing that all other donor suitability requirements are met. The decision of whether an initially reactive donor is to be reevaluated is left to the blood establishment. Furthermore, a blood establishment may adopt more stringent procedures, provided they are consistent with these recommendations. At each step in this procedure, the performance of each test and the interpretation of results should be as specified in the package insert for that kit.

I. Test for HBsAg ("Screening Test")

Each donation shall be tested for HBsAg by an FDA-licensed test of third-generation sensitivity (21 CFR 610.40). [Footnote 2: Exceptions to each donation being tested for HBsAg may be made in the case of donations from "dedicated donors" as described in "Revised Guidelines for the Collection of Platelets, Pheresis," January 1985.]

- a. If the initial test result is non-reactive, the donor may be accepted and the unit may be used for transfusion or further manufacturing.
- b. If the initial test result is reactive, the sample should be retested, in duplicate.
 - i. If the sample is not repeatably reactive, the donor may be accepted and the unit may be used for transfusion or further manufacturing.
 - ii. If the sample is repeatably reactive (i.e., if either or both of the duplicates are reactive), the unit should not be used for transfusion or further manufacturing, except as provided in 21CFR 610.80(d). For evaluation of the donor, the sample should be tested by an FDA-licensed confirmatory (neutralization) test for HBsAg. See Section II.
 - iii. If repeat testing is inconclusive, not done in duplicate, initially misinterpreted, etc., see Section V and treat the unit as reactive.

II. Confirmatory Test for HBsAg

- a. If the sample is confirmed positive, the donor is permanently deferred.
- b. If the sample is not neutralizable, it should be tested for anti-HBc with an FDA-approved test. See Section III.
- c. If testing is inconclusive, see Section V and treat the unit as reactive.

III. Test for Anti-HBc

The testing referred to in this Section is intended to evaluate further the status of the donor. If this testing has already been performed in the course of surrogate testing for this donation, those results may be used.

- a. If the initial test result is reactive, the test should be repeated in duplicate with that kit.
 - i. If the sample is repeatably reactive (i.e., if either or both of the duplicates are reactive), the donor should be permanently deferred.
 - ii. If the sample is not repeatably reactive, proceed as in Section III.b, below.
- b. If the sample is non-reactive, the donor need not be excluded but should be put in a status that will require review (see Section IV) at the time of the subsequent donation, after a minimum of eight weeks.
- c. If testing is inconclusive, not done in duplicate, initially misinterpreted, etc., see Section V and treat the unit as reactive.

IV. Tests at the Time of the Subsequent Donation

A minimum of eight weeks should elapse between the previous donation (i.e., that considered in Sections I-III) and the one considered in this Section. A sample drawn at the time of this subsequent donation should be initially tested and, if reactive, retested in duplicate for HBsAg and for anti-HBc as indicated in Sections I and III, respectively, above.

- a. If the sample drawn at this time is reactive for anti-HBc (regardless of the result of the test for HBsAg), the donor should be permanently deferred and the unit should not be used.
- b. If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is non-reactive for HBsAg, the donor may be accepted and the unit may be used.
- c. If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is reactive for HBsAg, the sample should be subjected to the testing algorithm which begins with I.b, above. [Footnote 3: If this testing leads to point III.a.ii or III.b, above, i.e., if the sample is repeatably reactive for HBsAg but not neutralizable in the confirmatory test for HBsAg and non-reactive for anti-HBc, the unit should not be used but the donor may (at his/her own discretion and that of the establishment) return and re-enter the testing scheme at point IV, above. Such returns may be repeated (at the discretion of the donor and the establishment) until a final decision (i.e., that equivalent to IV.a or IV.b) can be made on the basis of the testing algorithm. (It is recommended that, for the initial and the second donations, the tests for HBsAg be performed with one type of kit -- e.g., mouse monoclonal EIA from the same manufacturer. If the results suggest that the samples are repeatably reactive but not confirmed because of interaction with animal antibody used in this kit, a kit employing antibody from a different species may be used to test subsequent donations.)

V. If, at any point after an initially reactive result for HBsAg but before completion of the testing on a donation, the quantity of material from that donation becomes insufficient for completion of the testing, the unit must not be used and the testing algorithm which begins with IV, above, should be used in determining donor acceptability.

Appendix B - Original HBsAg specification in XML

```
<document>
  <section title = "Tests" meta = "object">
    <section title = "(I) HBsAg test">
      <sentence meta = "head">
        test the sample for HBsAg.
      </sentence>
      <section title = "(a) Non-reactive">
        <sentence>
          If the initial test result is non-reactive for HBsAg, the donor may be accepted and the unit
          may be used for transfusion or further manufacturing.
        </sentence>
      </section>
      <section title = "(b) Reactive">
        <sentence>
          If the initial test result is reactive for HBsAg, the sample should be retested, in duplicate.
        </sentence>
        <section title = "Repeated test" meta = "sub">
          <sentence>
            If the sample is not repeatably reactive for HBsAg, the donor may be accepted and the unit
            may be used for transfusion or further manufacturing.
          </sentence>
          <sentence>
            If the sample is repeatably reactive for HBsAg(i.e., if either or both of the duplicates
            are reactive), the unit should not be used for transfusion or further manufacturing, except
            as provided in '21 CFR 610.80(d)'.
          </sentence>
          <sentence meta = "sub">
            For evaluation of the donor, the sample should be tested by an FDA-licensed confirmatory
            (neutralization) test for HBsAg (See Section II).
          </sentence>
          <sentence>
            If repeat testing is inconclusive, not done in duplicate, or initially misinterpreted,
            see Section V and treat the unit as reactive for HBsAg.
          </sentence>
        </section>
      </section>
    </section>
    <section title = "(II) Confirmatory test">
      <sentence meta = "head">
        test the sample by an FDA-licensed confirmatory test for HBsAg.
      </sentence>
      <section title = "(a) Positive">
        <sentence>
          If the sample is confirmed positive, the donor is permanently deferred.
        </sentence>
      </section>
      <section title = "(b) Not neutralizable">
        <sentence>
          If the sample is not neutralizable, it should be tested for anti-HBc with an FDA-approved test
          (See Section III).
        </sentence>
      </section>
      <section title = "(c) Inconclusive">
        <sentence>
          If testing is inconclusive, see Section V and treat the unit as reactive for HBsAg.
        </sentence>
      </section>
    </section>
    <section title = "(III) Anti-HBc test">
      <sentence meta = "head">
        test the sample for anti-HBc with FDA-approved test.
      </sentence>
      <section title = "Intro">
        <sentence meta = "comment">
          The testing referred to in this Section is intended to evaluate further the status of the donor.
        </sentence>
      </section>
    </section>
  </document>
```

```

</sentence>
<sentence>
  If this testing has already been performed in the course of surrogate testing for this donation,
  those results may be used.
</sentence>
</section>
<section title = "(a) Reactive">
  <sentence>
    If the initial test result is reactive for anti-HBc, the test should be repeated in duplicate
    with that kit.
  </sentence>
  <section title = "Repeated test" meta = "sub">
    <sentence>
      If the sample is repeatably reactive for anti-HBc (i.e., if either or both of the
      duplicates are reactive), the donor should be permanently deferred.
    </sentence>
    <sentence>
      If the sample is not repeatably reactive for anti-HBc, proceed as in Section III.b, below.
    </sentence>
  </section>
</section>
<section title = "(b) Non-reactive">
  <sentence>
    If the sample is non-reactive for anti-HBc, the donor need not be excluded, but should be put in a
    status, and then will require review (see Section IV) at the time of the subsequent donation,
    after a minimum of eight weeks.
  </sentence>
</section>
<section title = "(c) Inconclusive">
  <sentence>
    If testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V
    and treat the unit as reactive for Anti-HBc.
  </sentence>
</section>
</section>
<section title = "(IV) Tests at the time of the subsequent donation">
  <sentence meta = "head">
    test subsequent donation.
  </sentence>
  <section title = "Intro">
    <sentence>
      A minimum of eight weeks should elapse between the previous donation (i.e., that considered
      in Sections I-III) and the one considered in this Section.
    </sentence>
    <sentence>
      A sample drawn at the time of this subsequent donation should be initially tested and, if it is
      reactive for HBsAg, it should be retested in duplicate for HBsAg and for anti-HBc as indicated
      in Sections I and III, respectively, above.
    </sentence>
  </section>
  <section title = "(a) Reactive">
    <sentence>
      If the sample drawn at this time is reactive for anti-HBc (regardless of the result of the test
      for HBsAg), the donor should be permanently deferred and the unit should not be used.
    </sentence>
  </section>
  <section title = "(b) Non-reactive">
    <sentence>
      If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is
      non-reactive for HBsAg, the donor may be accepted and the unit may be used.
    </sentence>
  </section>
  <section title = "(c) Non-reactive for anti-HBc and the initial test result is reactive for HBsAg">
    <sentence>
      If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is reactive
      for HBsAg, the sample should be subjected to the testing algorithm which begins with I.b, above.
    </sentence>
  </section>
</section>

```

```
</section>
<section title = "(V) inconclusive case" meta = "head">
  <sentence meta = "head">
    check for an inconclusive case.
  </sentence>
  <sentence>
    If the quantity of material from that donation becomes insufficient for completion of the testing
    at any point after an initially reactive result for HBsAg but before completion of the testing
    on a donation, the unit must not be used and the testing algorithm which begins with Section IV
    should be used in determining donor acceptability.
  </sentence>
</section>
</section>
</document>
```

Appendix C - Original HBsAg specification in XML snapshot

Tree	Meta info
- Requirements	
- Tests	object
- (I) HBsAg test	
- test the sample for HBsAg.	head
- Each donation shall be tested for HBsAg by an FDA-licensed test for third-generation sensitivity.	
- (a) If the initial test result is non-reactive for HBsAg, the donor may be accepted and the unit may be used...	
- (b) If the initial test result is reactive for HBsAg, the sample should be retested for HBsAg, in duplicate.	otherwise
- Repeated test	sub
- If the sample is not repeatably reactive for HBsAg, the donor may be accepted and the unit may be use...	
- If the sample is repeatably reactive for HBsAg(i.e., if either or both of the duplicates are reactive), the u...	otherwise
- For evaluation of the donor, the sample should be tested by an FDA-licensed confirmatory (neutralizati...	sub
- If repeat testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and tre...	
- (II) Confirmatory test	
- test the sample by an FDA-licensed confirmatory test for HBsAg.	head
- (a) If the sample is confirmed positive, the donor is permanently deferred.	
- (b) If the sample is not neutralizable, it should be tested for anti-HBc with an FDA-approved test (See Sect...	otherwise
- (c) If testing is inconclusive, see Section V and treat the unit as reactive for HBsAg.	
- (III) Anti-HBc test	
- test the sample for anti-HBc with FDA-approved test.	head
- Intro	
- The testing referred to in this Section is intended to evaluate further the status of the donor.	comment
- If this testing has already been performed in the course of surrogate testing for this donation, those res...	
- (a) If the initial test result is reactive for anti-HBc, the test should be repeated in duplicate with that kit.	
- Repeated test	sub
- If the sample is repeatably reactive for anti-HBc (i.e., if either or both of the duplicates are reactive), t...	
- If the sample is not repeatably reactive for anti-HBc, proceed as in Section III.b, below.	otherwise
- (b) If the sample is non-reactive for anti-HBc, the donor need not be excluded, but should be put in a st...	otherwise
- (c) If testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and treat th...	
- (IV) Tests at the time of the subsequent donation	
- review the donor at time of subsequent donation.	head
- Intro	
- A minimum of eight weeks should elapse between the previous donation (i.e., that considered in Sectio...	
- A sample drawn at the time of this subsequent donation should be initially tested and, if it is reactive for...	
- (a) Reactive	
- (a) If the sample drawn at this time is reactive for anti-HBc (regardless of the result of the test for HBsA...	
- (b) If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is non-reactive f...	otherwise
- (c) If the sample drawn at this time is non-reactive for anti-HBc and the initial test result is reactive for H...	otherwise
- (V) inconclusive case	
- check for an inconclusive case.	head
- If the quantity of material from that donation becomes insufficient for completion of the testing at any point ...	

Appendix D - Original HBsAg specification in TLG

```
class Tests.

Donor :: Undefined.
Unit  :: Undefined.
Sample :: String.
Subsequent_Sample :: String.

test Sample for HBsAg :
  test Sample for HBsAg by FDA-licensed test for third-generation sensitivity,
  if Sample = "not reactive for HBsAg" then
    accept Donor ,
    use Unit for transfusion or further manufacturing
  else if Sample = "reactive for HBsAg" then
    retest Sample for HBsAg in duplicate,
    if Sample == "not reactive for HBsAg" then
      accept Donor ,
      use Unit for transfusion or further manufacturing
    else if Sample = "reactive for HBsAg" then
      not use Unit for transfusion or further manufacturing except as provided in 21_CfR_610_80d,
      test Sample by FDA-licensed confirmatory test for HBsAg
    endif,
    if testing be inconclusive or testing not be done in duplicate or testing be misinterpreted then
      check for inconclusive case,
      Sample := "reactive for HBsAg"
    endif
  endif.

test Sample by FDA-licensed confirmatory test for HBsAg :
  if Sample = "positive" then
    defer Donor
  else if Sample = "not neutralizable" then
    test Sample for anti-HBc with FDA-approved test
  endif,
  if testing be inconclusive then
    check for inconclusive case,
    Sample := "reactive for HBsAg"
  endif.

test Sample for anti-HBc with FDA-approved test :
  if this testing be performed in course of surrogate testing for this donation then
    use those results
  endif,
  if Sample = "reactive for anti-HBc" then
    retest Sample for anti-HBc in duplicate,
    if Sample = "reactive for anti-HBc" then
      defer Donor
    else if Sample = "not reactive for anti-HBc" then
      not exclude Donor ,
      put Donor in status,
      review Donor at time of subsequent donation
    endif
  else if Sample = "not reactive for anti-HBc" then
    not exclude Donor ,
    put Donor in status,
    review Donor at time of subsequent donation
  endif,
  if testing be inconclusive or testing not be done in duplicate or testing be misinterpreted then
    check for inconclusive case,
    Sample := "reactive for Anti-HBc"
  endif.

review Donor at time of subsequent donation :
  minimum of eight weeks elapse between previous donation and the subsequent donation,
  test Subsequent_Sample for HBsAg,
  if Subsequent_Sample = "reactive for anti-HBc" then
```

```
    defer Donor ,
      not use Unit
    else if Subsequent_Sample = "not reactive for anti-HBc" and Sample = "not reactive for HBsAg" then
      accept Donor ,
      use Unit
    else if Subsequent_Sample = "not reactive for anti-HBc" and Sample = "reactive for HBsAg" then
      test Subsequent_Sample for HBsAg
    endif.

  check for inconclusive case :
    if quantity of material become insufficient for completion of testing then
      not use Unit
    endif.

end class.
```

Appendix E - Modified HBsAg specification in XML

```
<document>
  <section title = "Tests" meta = "object">
    <section title = "Main test">
      <sentence meta = "head">
        test the donor.
      </sentence>
      <sentence>
        Assign the sample of the donor to the sample.
      </sentence>
      <sentence>
        (I) The sample should be tested for HBsAg.
      </sentence>
      <sentence>
        (a,b) If the sample is non-reactive for HBsAg or the sample is not repeatably reactive for HBsAg,
        the donor may be accepted and the unit may be used for transfusion or further manufacturing.
      </sentence>
      <sentence meta = "otherwise">
        If the sample is repeatably reactive for HBsAg(i.e., if either or both of the duplicates are reactive),
        the unit should not be used for transfusion or further manufacturing, except as provided in
        "21 CFR 610.80(d)".
      </sentence>
      <section title = "(II) confirmatory test" meta = "sub">
        <sentence>
          The sample should be tested for confirmatory HBsAg.
        </sentence>
        <sentence>
          (a) If the sample is confirmed positive, the donor is permanently deferred.
        </sentence>
        <sentence meta = "otherwise">
          (b) If the sample is not neutralizable, the sample should be tested for anti-HBc with an FDA-approved
          test (See Section III).
        </sentence>
      </section>
      <section title = "(III) Anti-HBc test">
        <sentence meta = "sub">
          If the sample is repeatably reactive for anti-HBc (i.e., if either or both of the duplicates
          are reactive), the donor should be permanently deferred.
        </sentence>
        <sentence meta = "otherwise">
          (b) If the sample is not repeatably reactive for anti-HBc or the sample is non-reactive for anti-HBc,
          the donor need not be excluded, but should be put in a status, and then will require review
          (see Section IV) at the time of the subsequent donation, after a minimum of eight weeks.
        </sentence>
      </section>
      <section title = "(IV) Test at the time of the subsequent donation" meta = "sub">
        <section title = "Intro">
          <sentence>
            A minimum of eight weeks should elapse between the previous donation (i.e., that considered in
            Sections I-III) and the one considered in this Section.
          </sentence>
          <sentence>
            Assign the subsequent sample of the donor to the subsequent sample.
          </sentence>
          <sentence>
            The subsequent sample should be tested for HBsAg.
          </sentence>
          <sentence>
            If the subsequent sample is repeatably reactive for HBsAg, the subsequent sample should be
            tested for confirmatory HBsAg.
          </sentence>
          <sentence meta = "sub">
            If the subsequent sample is not neutralizable, the subsequent sample should be tested for
            anti-HBc with an FDA-approved test.
          </sentence>
        </section>
      </section>
    </section>
  </document>
```

(a) If the subsequent sample is reactive for anti-HBc (regardless of the result of the test for HBsAg), the donor should be permanently deferred and the unit should not be used.

</sentence>

<sentence meta = "otherwise">

(b) If the subsequent sample is non-reactive for anti-HBc and the initial test result is non-reactive for HBsAg, the donor may be accepted and the unit may be used.

</sentence>

<sentence meta = "otherwise">

(c) If the subsequent sample is non-reactive for anti-HBc and the initial test result is reactive for HBsAg, the sample should be subjected to the testing algorithm which begins with I.b, above.

</sentence>

</section>

</section>

</section>

</section>

</section>

<section title = "HBsAg test">

<sentence meta = "head">

Test the sample for HBsAg.

</sentence>

<sentence>

Each donation shall be tested for HBsAg by an FDA-licensed test for third-generation sensitivity.

</sentence>

<sentence>

If the sample is reactive for HBsAg, the sample should be retested for HBsAg, in duplicate.

</sentence>

<sentence>

If repeat testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and treat the unit as reactive for HBsAg.

</sentence>

</section>

<section title = "Confirmatory test">

<sentence meta = "head">

Test the sample for confirmatory HBsAg.

</sentence>

<sentence>

For evaluation of the donor, the sample should be tested by an FDA-licensed confirmatory (neutralization) test for HBsAg (See Section II).

</sentence>

<sentence>

If testing is inconclusive, see Section V and treat the unit as reactive for HBsAg.

</sentence>

</section>

<section title = "Anti-HBc test">

<sentence meta = "head">

test the sample for anti-HBc with FDA-approved test.

</sentence>

<section title = "Intro">

<sentence meta = "comment">

The testing referred to in this Section is intended to evaluate further the status of the donor.

</sentence>

<sentence>

If this testing has already been performed in the course of surrogate testing for this donation, those results may be used.

</sentence>

</section>

<section>

<sentence>

If the sample is reactive for anti-HBc, the sample should be retested for anti-HBc in duplicate with that kit.

</sentence>

<sentence>

If testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and treat the unit as reactive for Anti-HBc.

</sentence>

</section>

</section>

```
<section title = "(V) inconclusive case">
  <sentence meta = "head">
    check for an inconclusive case.
  </sentence>
  <sentence>
    If the quantity of material from that donation becomes insufficient for completion of the testing
    at any point after an initially reactive result for HBsAg but before completion of the testing
    on a donation, the unit must not be used and the testing algorithm which begins with Section IV
    should be used in determining donor acceptability.
  </sentence>
</section>
</section>
</document>
```

Appendix F - Modified HBsAg specification in XML snapshot

Tree	Meta info
- Requirements	
- Tests	object
- Main test	
- test the donor.	head
- Assign the sample of the donor to the sample.	
- (I) The sample should be tested for HBsAg.	
- (a,b) If the sample is non-reactive for HBsAg or the sample is not repeatably reactive for HBsAg, the don...	
- If the sample is repeatably reactive for HBsAg(i.e., if either or both of the duplicates are reactive), the unit...	otherwise
- (II) confirmatory test	sub
- The sample should be tested for confirmatory HBsAg.	
- (a) If the sample is confirmed positive, the donor is permanently deferred.	
- (b) If the sample is not neutralizable, the sample should be tested for anti-HBc with an FDA-approved te...	otherwise
- (III) Anti-HBc test	
- If the sample is repeatably reactive for anti-HBc (i.e., if either or both of the duplicates are reactive), t...	sub
- (b) If the sample is not repeatably reactive for anti-HBc or the sample is non-reactive for anti-HBc, th...	otherwise
- (IV) Test at the time of the subsequent donation	sub
- Intro	
- A minimum of eight weeks should elapse between the previous donation (i.e., that considered in ...	
- Assign the subsequent sample of the donor to the subsequent sample.	
- The subsequent sample should be tested for HBsAg.	
- If the subsequent sample is repeatably reactive for HBsAg, the subsequent sample should be te...	
- If the subsequent sample is not neutralizable, the subsequent sample should be tested for anti-...	sub
- (a) If the subsequent sample is reactive for anti-HBc (regardless of the result of the test for HBs...	
- (b) If the subsequent sample is non-reactive for anti-HBc and the initial test result is non-reactive...	otherwise
- (c) If the subsequent sample is non-reactive for anti-HBc and the initial test result is reactive for ...	otherwise
- HBsAg test	
- Test the sample for HBsAg.	head
- Each donation shall be tested for HBsAg by an FDA-licensed test for third-generation sensitivity.	
- If the sample is reactive for HBsAg, the sample should be retested for HBsAg, in duplicate.	
- If repeat testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and treat t...	
- Confirmatory test	
- Test the sample for confirmatory HBsAg.	head
- For evaluation of the donor, the sample should be tested by an FDA-licensed confirmatory (neutralization)...	
- If testing is inconclusive, see Section V and treat the unit as reactive for HBsAg.	
- Anti-HBc test	
- test the sample for anti-HBc with FDA-approved test.	head
- Intro	
- The testing referred to in this Section is intended to evaluate further the status of the donor.	comment
- If this testing has already been performed in the course of surrogate testing for this donation, those res...	
- If the sample is reactive for anti-HBc, the sample should be retested for anti-HBc in duplicate with that kit.	
- If testing is inconclusive, not done in duplicate, or initially misinterpreted, see Section V and treat the u...	
- (V) inconclusive case	
- check for an inconclusive case.	head
- If the quantity of material from that donation becomes insufficient for completion of the testing at any point ...	

Appendix G - Modified HBsAg specification in TLG

```
class Tests.

Sample :: String.
Subsequent_Sample :: String.

test Donor :
  Sample := Sample of Donor,
  test Sample for HBsAg returning Sample,
  if Sample = "not reactive for HBsAg" or Sample = "not repeatably reactive for HBsAg" then
    accept Donor ,
    use Unit of Donor for transfusion or further manufacturing
  else if Sample = "repeatably reactive for HBsAg" then
    not use Unit of Donor for transfusion or further manufacturing except as provided in 21_CfR_610_80d,
    test Sample for confirmatory HBsAg returning Sample,
    if Sample = "positive" then
      defer Donor
    else if Sample = "not neutralizable" then
      test Sample for anti-HBc returning Sample,
      if Sample = "repeatably reactive for anti-HBc" then
        defer Donor
      endif
    else if Sample = "not repeatably reactive for anti-HBc" or Sample = "not reactive for anti-HBc" then
      not exclude Donor ,
      put Donor in status,
      review Donor at time of subsequent donation,
      minimum of eight weeks elapse between previous donation and the subsequent donation,
      Subsequent_Sample := Subsequent_Sample of Donor,
      test Subsequent_Sample for HBsAg returning Subsequent_Sample,
      if Subsequent_Sample = "repeatably reactive for HBsAg" then
        test Subsequent_Sample for confirmatory HBsAg returning Subsequent_Sample,
        if Subsequent_Sample = "not neutralizable" then
          test Subsequent_Sample for anti-HBc returning Subsequent_Sample
        endif
      endif,
      if Subsequent_Sample = "reactive for anti-HBc" then
        defer Donor ,
        not use Unit of Donor
      else if Subsequent_Sample = "not reactive for anti-HBc" and Sample = "not reactive for HBsAg" then
        accept Donor ,
        use Unit of Donor
      else if Subsequent_Sample = "not reactive for anti-HBc" and Sample = "reactive for HBsAg" then
        test Subsequent_Sample for HBsAg returning Subsequent_Sample
      endif
    endif
  endif.

test Sample for HBsAg returning Sample :
  test Sample for HBsAg by FDA-licensed test for third-generation sensitivity,
  if Sample = "reactive for HBsAg" then
    retest Sample for HBsAg in duplicate
  endif,
  if testing be inconclusive or testing not be done in duplicate or testing be misinterpreted then
    check for inconclusive case,
    Sample := "reactive for HBsAg"
  endif.

test Sample for confirmatory HBsAg returning Sample :
  test Sample by FDA-licensed confirmatory test for HBsAg,
  if testing be inconclusive then
    check for inconclusive case,
    Sample := "reactive for HBsAg"
  endif.

test Sample for anti-HBc returning Sample :
  if this testing be performed in course of surrogate testing for previous donation then
```

```
        use those results
    endif,
    if Sample = "reactive for anti-HBc" then
        retest Sample for anti-HBc in duplicate
    endif,
    if testing be inconclusive or testing not be done in duplicate or testing be misinterpreted then
        check for inconclusive case,
        Sample := "reactive for Anti-HBc"
    endif.

check for inconclusive case :
    if quantity of material become insufficient for completion of testing then
        not use Unit of Donor
    endif.

end class.

class Donor.

    Unit :: Undefined.
    Sample :: String.
    Subsequent_Sample :: String.

end class.
```