Toward the Discovery and Extraction of Money Laundering Evidence from Arbitrary Data Formats using Combinatory Reductions

Alonza Mumford, Duminda Wijesekera

George Mason University

amumford@gmu.edu, dwijesek@gmu.edu

November 19, 2014

Overview

1. Problem, Background & Architecture
2. Motivating Example
3. Descriptive Data Modeling with DFDL
4. Ontological Modeling with XLink
5. Compiler Specification in CRSX
6. Future Work

Problem Statement

—The "wild, wild west analogy" is applicable to the variety of non-standardized, structured and unstructured data that exist in public and private domains

—This phenomena disrupts the opportunity to "get to" this heterogeneous raw data in a generic manner, represent the connections between this data that is distributed, and make new discovers based partially on those connections.

—The manner in which we express these connections must contextualize specific data with a conceptual understanding of a problem, investigation or inquiry without having to change or alter the data itself

Background (Challenges)

Challenge 1

To provide a mechanism that can be used to describe and access any number of data formats.

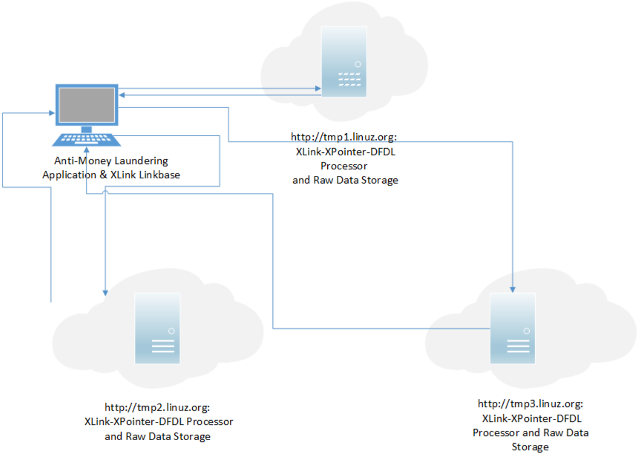
Challenge 2

To provide a lightweight, metadata-based discovery and extraction of arbitrary data fragments from raw data stores not co-located with each other. The intent is to avoid some of the system development and maintenance costs associated with major data conversion, and database storage and indexing.

Challenge 3

To provide a mechanism that incorporates ontological engineering for describing the meaning of data fragments (i.e., parts of data) within the context of a particular domain of understanding, inquiry or investigation.

Architecture (XLink-DFDL Processing)



Anti-Money Laundering
Application & XLink Linkbase

http://tmp1.linuz.org:
XLink-XPointer-DFDL
Processor
and Raw Data Storage

http://tmp2.linuz.org:
XLink-XPointer-DFDL Processor
and Raw Data Storage

http://tmp3.linuz.org:
XLink-XPointer-DFDL
Processor and Raw Data
Storage

Motivating Example: Analysis of a Money Laundering Scheme

Step 1: Placement Stage

A number of policies entered into by the same insurer (i.e. a person or company that underwrites an insurance risk) for small amounts and then canceled early at the same time [1, 2].

Step 2: Layering Stage

Requests for return premiums in currencies different to the original premium or requests for return premiums to an account different from the original account [1, 2].

Step 3: Integration Stage

Return premium being credited in different currency or different account, which represents launder money [1, 2].

Example (part 1): Inspection of a Policy Account Record

```
(a.1) Input "policy account record" data:

PLCYACC/741032-1071//
DATE/2013-09-28//
PLCYHLD/Allegier, Cox & Associates, Inc.//
INSUR/ALI Corp.//
PAYER/Grupo Palermo S.A.//
PAYCUR/Peso (ARG)//
PRMAMT/42004.98//

(a.2) DFDL generated XML model:

<policyAccountRecord>
    <policyAccountIdentifier>741032-1071</
         policyAccountIdentifier>
    <policyStartDate>2013-09-28</policyStartDate>
    <policyHolder>Allegier, Cox & Associates Inc.</
         policyHolder>
    <policyInsurer>ALI Corp.</policyInsurer>
    <payerName>Grupo Palermo S.A.</payerName>
    <payerCurrency>Peso (Argentine)</payerCurrency>
    <premiumAmount>42004.98</premiumAmount>
</policyAccountRecord>
```

Example (part 2): Inspection of a Credit Request Record

```
(b.1) Input "credit request record" data:

 [[[[[[[CRDREQ%]741032-1071%]
CRDATE%]2013-11-02%]
PAYEE%]Allegier, Cox & Associates, Inc.%]
PAYCUR%]USD%]
CRDAMT%]5000.00%]

(b.2) DFDL generated XML model:

<creditRequestRecord>
    <creditRequestIdentifier>741032-1071</creditRequestIdentifier>
    <creditRequestDate>2013-11-02</creditRequestDate>
    <payeeName>Allegier, Cox & Associates, Inc.</payeeName>
    <payeeCurrency>USD</payeeCurrency>
    <creditAmount>5000.00</creditAmount>
</creditRequestRecord>
```

Descriptive Data Modeling with DFDL

—Data Format Description Language [3] is not a data-format... it's a way to describe any data format (i.e., dense binary and text)

—Prescriptive standards (e.g. ASN.1, JSON) provides a structure for data as certain people think it should be in lieu of the structure of data as it actually is

—DFDL's syntax uses a subset of XML Schema with DFDL annotations to represent the physical properties of data

—As of January 2011, DFDL v1 is a "Recommendation" standard of the Open Grid Forum (OGF) designed to "enable efficient representation of data... essential to high performance data exchange in grid computing and... distributed computing."

Modeling of Logical Datatypes & Constraints

Logical Datatypes & Constraints	**Context Free Grammar (CFG) example**	**Higher-Order Abstract Syntax (HOAS) example**
Structures (xs:complexType)	2 <dcl_xs> ::= "<" XS_COMPONENT <stmt>* "</" XS_COMPONENT ">" 11 XS_COMPONENT ::= "xs:complexType"	XsComponent[ComplexType]
Atomic data values (xs:simpleType)	11 XS_COMPONENT ::= "xs:simpleType"	XsComponent[SimpleType]
Ordering (xs:sequence or xs:choice)	11 XS_COMPONENT ::= "xs:sequence" | "xs:choice"	XsComponent[Sequence], XsComponent[Choice]
Occurences (xs:minOccurs or xs:maxOccurs)	5 <stmt> ::= XS_ATTRIBUTE "=" <xs_attribute_value> 16 XS_ATTRIBUTE ::= "xs:minOccurs" | "xs:maxOccurs" 8 <xs_attribute_value> ::= <xs_number>	ComponentAttribute[MinOccurs], ComponentAttribute[MaxOccurs]

Modeling of Physical Representations

DFDL Physical Representation Properties	Context Free Grammar (CFG) example	Higher-Order Abstract Syntax (HOAS) example
Physical types (dfdl:representation)	5 <stmt> ::= DFDL_ATTRIBUTE "=" <dfdl_attribute_value> 18 DFDL_ATTRIBUTE ::= "representation" 7 <dfdl_attribute_value> ::= <dfdl_enum_number>	FormatProperty[Represetation]
Delimiters (dfdl:initiator, dfdl:separator, dfdl:terminator)	18 DFDL_ATTRIBUTE ::= "initiator" 7 <dfdl_attribute_value> ::= <dfdl_string_value> | <reg_exp_value>	FormatProperty[Initator], FormatProperty[Separator], FormatProperty[Terminator
Extraction of elements (dfdl:lengthKind)	18 DFDL_ATTRIBUTE ::= "lengthKind" 7 <dfdl_attribute_value> ::= <dfdl_enum_value>	FormatProperty[LengthKind]
Points of uncertainty (dfdl:discriminator)	3 <dcl_dfdl> ::= "<" DFDL_ADMIN <stmt>* ">" <dcl_dfdl> "</" DFDL_ADMIN ">" 15 DFDL_ADMIN ::= "dfdl:discriminator"	DfdlValidation[Discriminator]
Detecting occurrences (dfdl:occursCount)	18 DFDL_ATTRIBUTE ::= "occursCount" 7 <dfdl_attribute_value> ::= <non_neg_int_value> | <dfdl_exp_value>	FormatProperty[OccursCount]

Example (part 3): Policy Account Record Schema

```
...
<xs:element name="policyAccountRecord" minOccurs="0" maxOccurs="unbounded" dfdl:lengthKind="implicit">
  <xs:complexType>
      <xs:sequence dfdl:sequenceKind="ordered">
        <annotation>
          <xs:appinfo source="http://www.ogf.org/dfdl/v1.0">
            <dfdl:element representation="text" encoding="ascii" lengthKind="delimited" sequenceKind="ordered" initiator="//" separator="/" separatorPosition="infix" separatorPolicy="required"/>
          </xs:appinfo>
        </annotation>
        <xs:element name="policyAccountIdentifier" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
        <xs:element name="policyStartDate" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
        <xs:element name="policyHolder" type="xs:string" dfdl:lengthKind="explicit" dfdl:length="20"/>
        <xs:element name="policyInsurer" type="xs:string" dfdl:lengthKind="implicit"/>
        <xs:element name="payerName" type="xs:string" dfdl:lengthKind="implicit"/>
        <xs:element name="payerCurrency" type="xs:string" dfdl:lengthKind="implicit"/>
        <xs:element name="premiumAmount" type="xs:string" dfdl:lengthKind="implicit"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...
```

Ontological Modeling with XLink

Use XLink-based ontological engineering and analytical reasoning to define the concepts relevant to the money laundering domain [4, 5]:

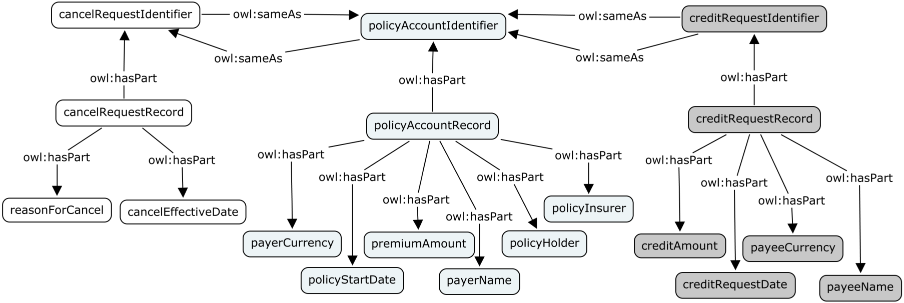
—A *concept* definition conveys the name of an evidentiary fact and its value data type

—A *conceptual ontology* of the "anti-money laundering" domain is given to show the kinds of classes and properties used in the domain

—*Classes* are identified by nodes and *properties* are identified by directed paths or arcs

—The *conceptual labels* associated with properties represent composition (i.e., hasPart) and equivalence (i.e., sameAs) relations between classes.

Example: Ontological Model of Money Launder Domain



Ontological Modeling with XLink

Concept	**Example**
Classes	propertyAccountRecord, cancelRequestRecord and creditRequestRecord (ref: figs. 5 and 8)
Instances	An instance of a propertyAccountRecord is one bearing "741032-1071" as the policyAccountIdentifier (ref: fig 2, a.1).
Relations: hasPart, sameAs	The three properties, policyAccountIdentifier, cancelRequestIdentifier, and creditRequestIdentifier are equivalent (sameAs) (ref. figs. 8 and 9).
Properties	policyAccountIdentifier, policyStartDate, policyHolder, policyInsurer are properties of a policyAccountRecord (ref: figs. 5, 8 and 9).
Values	"USD" and "5000.00" are the values of payeeCurrency and premiumAmount respectively for a particular instance of a creditRequestRecord (ref: fig. 2, b.2).
Rules	The three properties, policyAccountIdentifier, cancelRequestIdentifier, and creditRequestIdentifier are equivalent (sameAs) if they evaluate to the same value, for example, "741032-1071".

XLink-XPointer-based Concept Modeling

Attribute	Value	Description
xlink:type	extended	Parent element, which defines a complex link in which multiple links can be combined based on other attributes.
	resource	Child element of extended-Type element, which provides a local resource to participate in the link.
	locator	Child element of extended-Type element, which specifies the location of a remote resource participating in the link.
	arc	Child element of extended-Type element, which define traversal rules between the link's participating resources.
xlink:label	<string>	Traversal attribute of extended-, resource-Type elements, which provides a reference (of itself) to arc-Type in creating a traversal arc.
xlink:from, xlink:to	<string>	Traversal attributes of arc-Type element, which define the source and target resources of the arc link.

XLink-XPointer-based Concept Modeling

Attribute	Value	Description
xlink:href	URL	The linked URL
xlink:role	<string>	Semantic attribute of extended-, resource-Type elements, which indicates a property of the resourcein a computer readable-form.
xlink:arcrole	<string>	Semantic attribute of arc-Type element, which coincides with the [RDF] notion of a property, where "the role can be interpreted as stating that "starting-resource HAS arc-role ending-resource."
#xpointer		Creates XPointer fragment links with syntax:#xpointer(id(""))

Example (part 1): Policy Account Schema with XLink-XPointer Syntax

```
      ...
    <xs:element name="policyAccountRecord" minOccurs="0"
        maxOccurs="unbounded" dfdl:lengthKind="implicit"
        xlink:label="PolicyAccountRecord" xlink:href="http:
        //tmp1.linuz.org/policyAccountSchema.dfdl">
        ...
                <xs:element name="policyAccountIdentifier"
                    type="xs:string" dfdl:lengthKind="
                    explicit" dfdl:length="20" xlink:label=
                    "PolicyAccountIdentifier" xlink:type="
                    resource" xlink:href="http://linuz1/
                    policyAccountSchema.dfdl#xpointer(///
                    policyAccountIdentifier[@xs:string=
                    value])"/>
        ...
```

Example (part 2): Linkbase loads on extraction

```
<linkbase xmlns:xlink="http://www.w3.org/1999/xlink" xlink:linkbase="http://www.w3.org/1999/xlink/properties/linkbase">
   <link xlink:type="extended" xlink:title="moneyLaunderLinkbase">

          <!-- Linkbase loads on extraction request. -->
      <basesloaded>
      <startrsrc xlink:label="filter_spec" xlink:href="/local/filter_spec.xml#params" />
      <linkbase xlink:label="linkbase" xlink:href="/local/linkbase.xml" />
      <load xlink:from="filter_spec" xlink:to="linkbase" actuate="onRequest" />
      </basesloaded>

       ...
```

Example (part 3): Linkbase arcs

```
...

<!-- Arcs between linkbase and DFDL-data stores. -->
<invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase" xlink:from="linkbase" xlink:from="PolicyAccountRecord"/>
<invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase" xlink:from="linkbase" xlink:from="CancelRequestRecord"/>
<invokeStoreArc xlink:type="arc" xlink:arcrole="linkbase" xlink:from="linkbase" xlink:from="RefundRequestRecord"/>

...
```

Example (part 4): Linkbase locators

```
...

<!-- Locator elements. -->
<loc xlink:type="locator" xlink:label="PolicyAccountRecord" xlink:href="http://tmp1.linuz.org/policyAccountSchema.dfdl"/>
<loc xlink:type="locator" xlink:label="PolicyAccountIdentifier" xlink:href="http://tmp1.linuz.orglinuz1/policyAccountSchema.dfdl#xpointer(////policyAccountIdentifier[@xs:string=value])"/>
<loc xlink:type="locator" xlink:label="PolicyStartDate" xlink:href="http://tmp1.linuz.org/policyAccountSchema.dfdl#xpointer(////policyStartDate[@xs:date=value])"/>
...
```

Example (part 5): Linkbase relationship between insurance records

```
        ...

        <!-- Relationship between policy account, cancel
            request and refund request identifiers. -->
        <invokeIdArc xlink:type="arc" xlink:arcrole="
            owl:sameAs" xlink:from="PolicyAccountIdentifier"
             xlink:to="CancelRequestIdentifier"/>
        <invokeIdArc xlink:type="arc" xlink:arcrole="
            owl:sameAs" xlink:from="PolicyAccountIdentifier"
             xlink:to="RefundRequestIdentifier"/>
        <invokeIdArc xlink:type="arc" xlink:arcrole="
            owl:sameAs" xlink:from="CancelRequestIdentifier"
             xlink:to="PolicyAccountIdentifier"/>

        ...
```

Example (part 6): Linkbase relationship between parts

```
...
  <!-- Relationship between policy account record and its parts. -->
<invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyAccountIdentifier" />
<invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyStartDate" />
<invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyHolder" />
<invokeParArc xlink:type="arc" xlink:arcrole="owl:hasPart" xlink:from="PolicyAccountRecord" xlink:to= "PolicyInsurer" />
...
```

Compiler Specification in CRSX

—For syntactic analysis, construct a DFDL grammar as a set of recursive definitions. This effort entails creating a context free grammar (CFG) production rule for each code fragment in the DFDL schema language.

—For semantic analysis:

  —Devise the desired higher-order abstract syntax (HOAS) for DFDL. This also entails creating a HOAS code fragment for each CFG production rule.

  —Devise CRS transformation rules to address a multitude of issues such as DFDL scoping semantics. DFDL addresses scoping semantics through transformation rules by exploiting lambda lifting and dropping

What is CRSX?

—Rewriting is theory of stepwise or discrete transformations of objects

—Combinatory Reduction System (CRS) extends the format of first-order rewriting (TRS) with an operator for abstraction over variables

—A legal TRS cannot express and manipulate terms with bound variables,

e.g. untyped λ-calculus with a beta reduction,

$(\ \lambda x.M)N \rightarrow M[x{:=}N]$

e.g. a map function that applies a function to all elements of a list,

map(f,nil) → nil
map(f, cons(h,t)) → cons(f(h),map(f,t))

—Kristoffer Rose contributes CRSX, which implements Klop's CRS with extensions to support the writing of compilers

Example 1 —CRSX Rewriting Rule System

—Consider example of the rewriting rule system,

+[3,1]
PlusS: +[S[#1],#2] → S[+[#1,#2]];

—Consider the meaning of the previous syntax [6]:

—Consider the form: **name[options]:pattern** → **contraction**, where name should be a constructor and the pattern and contraction should be terms

—Capitalized words (e.g. Cons and Nil) , numbers, and most other symbols are **constructors**, which take an optional ordered or positional **parameter** list in immediately following [ ]s

—Each parameter is itself a **term**, and called a **subterm**

—Names containing # are special pattern variables, or **meta-variables**, that are used to match arbitrary subterms at the indicated position

Example 2 —CRSX Rewriting Rule System

—Consider example of the rewriting rule system,

Let[E1, x.E2]
Let[Copy[#1]]: Let[#1, x.#2[x]] → #2[#1]

—Consider the meaning of the previous syntax [6]:

—Uncapitalized words (e.g. x and foo) denote **variables**

—Variables are useful for defining binding constructs. It allows **explicit scoping** consider,

e.g. E ::= let x := E1 in E2

—Let[#1, x.#2[x]], where #1 is a **meta-variable** pattern and x.#2[x] is a **meta-application** pattern; the constructor must be Let; there are exactly two subterms; the second subterm under the binder can be anything wherein the variable matched by x may occur, and is matched by #₂ where we keep track of all the actual **occurrences** of the bound variable

Example (part 1): Syntax to CFG Production Rule for DFDL

```
<xs:annotation>
      <dfdl:defineEscapeScheme name= "myEscapeScheme">
         ...
         </dfdl:escapeScheme escapeCharacter= "/" />
         ...
   <dfdl:element representation="text" escapeSchemeRef="myEscapeScheme"
         ...
```

Figure: Consider a small fragment of the DFDL Language that allows us to construct a data description of an *escape character*, which is used to invoke an alternative interpretation on subsequent characters in a character sequence.

Modeling of Logical Datatypes & Constraints

Logical Datatypes & Constraints	**Context Free Grammar (CFG) example**	**Higher-Order Abstract Syntax (HOAS) example**
Structures (xs:complexType)	2 <dcl_xs> ::= "<" XS_COMPONENT <stmt>* "</" XS_COMPONENT ">" 11 XS_COMPONENT ::= "xs:complexType"	XsComponent[ComplexType]
Atomic data values (xs:simpleType)	11 XS_COMPONENT ::= "xs:simpleType"	XsComponent[SimpleType]
Ordering (xs:sequence or xs:choice)	11 XS_COMPONENT ::= "xs:sequence" | "xs:choice"	XsComponent[Sequence], XsComponent[Choice]
Occurences (xs:minOccurs or xs:maxOccurs)	5 <stmt> ::= XS_ATTRIBUTE "=" <xs_attribute_value> 16 XS_ATTRIBUTE ::= "xs:minOccurs" | "xs:maxOccurs" 8 <xs_attribute_value> ::= <xs_number>	ComponentAttribute[MinOccurs], ComponentAttribute[MaxOccurs]

Example (part 2): Syntax to CFG Production Rule for DFDL

```
2   <dcl_xs> ::= "<" XS_ADMIN <stmt>* ">" <dcl_xs>* "</"
     XS_ADMIN ">"  |  "<" XS_COMPONENT <stmt>* ">" <dcl_xs>
     * "</" XS_COMPONENT ">"

3   <dcl_dfdl>  ::= "<" DFDL_ADMIN <stmt>* ">" <dcl_dfdl> "
     </" DFDL_ADMIN ">"  |  "<" DFDL_COMPONENT <stmt>* ">"
     <dcl_dfdl> "</" DFDL_COMPONENT ">"  |  ...

5   <stmt>      ::= "escapeSchemeRef" = <NCName_value>  |
     "name" = <QName>  |  ...

15  DFDL_ADMIN  ::= "dfdl:defineEscapeScheme"  |  ...
```

Figure: The following production rules specify how these data descriptions are formed:

Example (part 3): CFG Production Rule to HOAS for DFDL

```
TERM ::=(
    Let[ VALUE, TYPE, x::VALUE . TERM ];
    Lam[ VALUE, TYPE, x::VALUE . TERM ];
    Context[ ];
    Element[KIND, $List[ATTRIBUTE], $List[DFDL_PROPERTY],
          $List[XLP_ATTRIBUTE], TERM];
    Pair[ TERM, TERM ];
    Nil;
    T;
    T-Attribute
    T-BuildSchema
    T-BuildElement
    XML-Visit[ XLink-XPointer ]
);
```

Figure: Consider our top level terms for the DFDL CRSX system after normalization. Consider our top level term for representing explicit scoping. The terms are written in the form of a higher-order abstract syntax.

Example (part 4): Using rewrite rules to define explicit scoping of attributes

```
XsComponent-Attribute[Copy[#QName]]
:
{#Env; #QName: ComponentAttribute[#kind]}
XML-Attribute[ #prefix , #QName, #Value , ok.#Continuation[ok]]
->
{#Env}
 Let[ #Value , a.{#Env} AddXsAssoc[#prefix , #QName, a , ok.#Continuation[ok]]
;
```

Figure: The following rewrite rule match "XML-Attribute" constructor with parameters and appends the entry to the the local environment:

Example (part 5): Using rewrite rules to define nested scopes of elements

```

<dfdl:discriminator ...> ... </dfdl:discriminator>
```

Figure: Consider the block scope a DFDL element

Example (part 6): Using rewrite rules to define nested scopes of elements

```
DfdlValidation-Start[Copy[#QName]]
:
{#Env; #QName: DfdlValidation[#kind]}
XML-Start[#prefix, #QName, ok prefix.#Continuation[ok, prefix]]
-->
{#Env; "T-BuildElement": Save-BuildElement[#QName]} #Continuation[OK, #QName]
;
...
```

Figure: The following rewrite rules "mark" the start-of block in the environment:

Example (part 7): Using rewrite rules to define nested scopes of elements

```
DfdlValidation-End
:
{#Env}
XML-End[ #prefix , ok.#Continuation[ok]]
-->
{#Env}
T-BuildSchema[OK, #QName, DfdlValidation[#kind], #content,
     ok.#[ok]]
```

Figure: The following rewrite rules "mark" the end-of block in the environment:

Example (part 8): Using rewrite rules to fold and unfold the DFDL program

```
1   <dcl_schema>    ::= "<" XS_SCHEMA <stmt>* ">" <dcl_xs>*
      "</" XS_SCHEMA">" $

3   <dcl_xs>        ::= "<" DFDL_COMPONENT <stmt>* ">" <dcl_xs>
       "</" DFDL_COMPONENT ">"
```

Figure: Consider the recursive definition of Schema element:

Example (part 9): Using rewrite rules to fold and unfold the DFDL program

```
Schema-End
:
{#Env}
XML-End[ #prefix , ok.#Continuation[ok]]
-->
{#Env}
T-BuildSchema[OK, #QName, Schema , #content , ok.#[ok]]
        . . .
-[Copy[#QName]]
:
{#Env}
T-BuildSchema[OK, #QName, Schema , #content , ok.#[ok]]
-->
Element[#QName, Schema , $[{#Env}Get ,"ATTRIBUTE" , () ] ,[
     $[{#Env}Get ,"DFDL_PROPERTY" , () ] , $[{#Env}Get , "
     CONTENT" , () ]  ]
;
```

Future Work

The future work includes:

—Specifying the transformation and evaluation of the DFDL/XLink/HOF specification into parser combinator form

—Investigating the operational semantics of the higher-order function (HOF) and linking abstractions in order to optimize distributed data extraction

—Generating comparative performance metrics

References

- F. A. T. Force. (2014 (accessed August 19, 2014)) Money laundering.
- I. A. of Insurance Supervisors, *Examples of Money Laundering and Suspicious Transactions Involving Insurance*. International Association of Insurance Supervisors, 2004. [Online]. Available: http://books.google.com/books?id=bSvoHAAACAAJ
- O. D. WG, S. M. Hanson, and A. W. Powell, "Data format description language (dfdl) v1. 0 specification."
- S. DeRose, E. Maler, D. Orchard, and N. Walsh, "Xml linking language (xlink) version 1.1, w3c recommendation 06 may 2010," 2010.
- P. Grosso, E. Maler, J. Marsh, and N. Walsh, "Xpointer framework," *W3c recommendation*, vol. 25, 2003.
- K. H. Rose, "Crsx-combinatory reduction systems with extensions," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 10. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

The End