シーケンシャルデータを厳密に処理するプログラムの簡略な作成法

スキーマ言語を用いたデータ構造･アプリケーションソフトウエア間インターフェースの記述 ~ 入力処理プログラムの自動生成

松崎 恵一[1]、馬場 肇[2,1] 、周東 晃四郎[1] 、三浦 昭[1]
(1-ISAS/JAXA, 2-早稲田大学)

はじめに

シーケンシャルデータの処理プログラムの作成方法の指針を示す。ここで、シーケンシャルデータとは

– ファイル、メモリ、信号線

などの媒体の上に、データ項目を順に並べたものである。

計算機や通信で取り扱うデータの多くはシーケンシャルデータである。衛星システム（衛星⇔地上⇔ユーザ）においても、テレメトリ･コマンドのパケットや分散した計算機システムなどにおいて、多種多様なシーケンシャルデータが登場する。また、搭載機器間の通信のシリアル化 (例えばref1) などその重要性が増してきている。

衛星などのよく管理された開発では、シーケンシャルデータの書式を図あるいは表を用いて仕様書に記載する。これは人が読むためのものである。これを処理プログラムに焼きなおすのは通例手作業であった。これは手間だし、しばしミスが混入してきた。さらに、図表は意味が曖昧な場合があり、この場合、仕様書は知っている人にしか理解できないものであった。

観測データの二次処理や、研究者のデータ解析など、流れの下流では、より多くの人が処理のプログラムを書く。この場合、仕様の作成に使える労力は小さい。メモ程度のものを書くこともあるが、端折ってプログラムそのもので代用することもある。こうしたメモも曖昧で人間しか読めない。

以下では、より厳密に、より少ない作業量で、シーケンシャルデータの処理プログラム作成する方法を示す。

Step 1.シーケンシャルデータの書式を記述するスキーマを作成する

スキーマ言語は、曖昧なくデータの書式を定義するものである。近年普及したXML形式 (これ自身シーケンシャルデータの書式の一種) では、標準なスキーマ言語が幾つか普及している。一方、より一般的なシーケンシャルデータでは、利用可能なスキーマ言語が、これまで事実上存在していなかった。そこで、我々はスキーマ言語の検討に着手した。

データは、使われることで意味を持つ。あるものはプログラムに取り込まれ、あるものはいったん別の書式に変換され下流に流される。その途中では、しばし意味のない単なる書式の変換が登場する。たとえば、あるデータの項目が、ファイルに書かれていようとプログラムから変数として見えていようと、データそのものを取り扱う側から見れば区別の必要がない。我々は、スキーマ言語の設計において、表現の多態性に極力対応するよう心がけている。シーケンシャルデータに対するスキーマ言語も、より多態な表現に対応するスキーマ言語一つに位置づけた。我々は、この一群のスキーマ言語全体を mappingSchema と呼んでいる。これまでに、簡便で広範な文法体系が出来つつある。

ここでは、例として、図1に示すようにデータ項目が列挙されたシーケンシャルデータを考える。リスト1にmappingSchemaの文法に従ったスキーマ、リスト2にこのスキーマに合致するデータ (インスタンス) の例を示す。

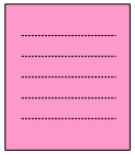

図1. シーケンシャルデータのイメージ
(図2との比較のために示した)

リスト1．シーケンシャルデータのスキーマの例

簡単のため、次節に示す空間の表記を省いている。

```
<?xml version="1.0" encoding="UTF-8"?>
<byte encode="txt">
  <group>
    <group>
      <byte length="6"><data type="string"/></byte>
      <byte length="1">
        <data type="string"><param name="pattern">[¥s]+</param></data>
      </byte>
      <data type="string"><param name="pattern">[a-zA-Z_][a-zA-Z0-9_]*</param></data>
      <value>(</value>
      <data type="string"><param name="pattern">[¥s]*</param></data>
      <data type="string"><param name="pattern">[0-9]+</param></data>
      <data type="string“><param name="pattern">[¥s]*</param></data>
      <value>)</value>
    </group>
    <zeroOrMore>
      <loopGroup>
        <data type="string"><param name="pattern">[¥s]+</param></data>
        <data type="string"><param name="pattern">[0-9]+</param></data>
      </loopGroup>
    </zeroOrMore>
    <choice>
      <choiceGroup>
        <data type="string"><param name="pattern">[¥s]+</param></data>
        <data type="string"><param name="pattern">#</param></data>
        <data type="string"><param name="pattern">.+</param></data>
      </choiceGroup>
      <choiceGroup>
      </choiceGroup>
    </choice>
  </group>
</byte>
```

はじめの 7byte が固定長のフィールドで、それ以降に可変長・可変個数のフィールドを持つテキスト形式。記号 # のあとコメントを記すことができる。
(この説明自身、自然言語による記述が曖昧なことを示す良い例になっている)

リスト2．リスト1のスキーマで規定されるシーケンシャルデータのインスタンスの例

(例1)

```
Vector a(3) 1 2 3 # position
```

(例2)

```
Vector b(2) 2 4
```

(例3)

```
Vector c(1) 1     # end
```

リスト1、リスト2の双方において
下線＋青で示す部分がこの書式が担うデータ項目
緑で示す部分がデータ項目の区切りを示す部分

mappingSchema

我々は、XML形式版のスキーマ が、既存で標準なスキーマ言語となるよう mappingSchema を設計した。こうすることで、先達の検討結果を踏まえ、最小限の労力で、言語を広げていくことができる。我々は、標準の XML スキーマ言語の中から、RELAX NG (ref 2) を mappingSchema の土台として選択した。これは、RELAX NG が純粋にXMLの書式を規定するためのものであり、単純かつ表現対象が広いからだ。

RELAX NG はあるスキーマを表記するのに、XMLを用いた表記法(XML Syntax)と、より簡素なテキスト形式で記述する表記法 (XML Syntax) が用意されている。我々は、現在、XMLを用いた表記法をもとに mappingSchema の文法を検討している。

シーケンシャルデータの書式では、項目の並べ方や、項目ごとの
- 表現方法 (テキスト表記やバイナリ表記など)
- フィールドの幅
- 型
- 値の範囲など

などを規定する必要がある (下線はバイナリに固有な項目)。

テキスト形式に対するスキーマ言語に、RELAX NG の文法を超える新たな概念を持ち込む必要はない。RELAX NG が XML 形式のスキーマ言語であるために存在する幾つかの制約をはずせばよい。これは、XMLファイルから要素や属性の指定を取り除けば、単なるテキスト形式のファイルになることに対応している。

バイナリ形式に対しては、下線で示した部分の概念を表す方法が必要である。RELAX NGでは、値をつめる要素や属性に対応して、elementや attribute という入れ物が用意されている。これに対応して、我々は バイナリ形式向けに bit や byte という入れ物を用意した。このような入れ物を mappingSchema では **container** と総称する。あるデータ形式に対する mappingSchema は、その形式固有な **container** を持つことができる。

container は、それぞれ固有な **attribute** を持つことができる。mappingSchema のXMLを用いた表記では、**attribute** をXML要素の属性を用いて記述する。XML形式の**container** である、要素や属性は名前 (name) という **attribute** が必須である。バイナリデータを保持する **container** である bit や byte は長さ(length) という**attribute** を持つことができる。

mappingSchema は**container** などの **schema element** の組み合わせで、構成される。XMLを用いた表記法では、一つの **schema element** を一つのXML要素で記述し、その種類を要素の名前で表現する。mappingSchema は、RELAX NG同様に、データ項目が並んでいることを **schema element** を並べることで表記する。あるデータ形式の **container** は、同じ種類あるいは異なる種類の **container** を、内部に保持することが出来る。このような入れ子の構造をXMLを用いた表記法ではXML要素の入れ子として表現する。どのような入れ子が許されるかは、データ形式自身で決まっている。bit や byteという **container** は任意の入れ子を組むことが出来る。

複雑なデータを表すには、条件的あるいは選択的、繰り返しに登場するといったことを表現する必要がでてくる。正規表現はこれらを ?, *, +, {m,n}, | といった記号で表記する (それぞれ、場合によって登場、0個以上の繰り返し、1個以上の繰り返し、m個以上、n個以下の繰り返し、いずれかが登場といった概念を記述できる)。mappingSchema の XML を用いた表記法では、これらを optional、zeroOrMore、oneOrMore、repeat、choice といった **schema element** で囲むことで表現する。repeat 以外は RELAX NG から受け継いだものである。これらは、どのような形式においても、共通的に用いることができる。ただし、実現可能な **container** の並べ方は、形式ごとの制約を受ける。シーケンシャルデータの**container** には、そのような制約がないので、自由に入れ子をつくることが出来る。

Step 2. マッピング記法を用い、シーケンシャルデータをどう受けとるか、ソフトウエアの内部インターフェースを記述する

シーケンシャルデータの生成･解釈はプログラム言語を用いて記述できる。プログラム全体の中で、この処理は外部仕様から直接影響をうける部分であり、その他データを本質的にいじる部分とは分けておくと、プログラム作成上都合がよい。両者の切り分けるには、変数や関数呼び出しなどソフトウエア内部での受け渡し方法といったプログラム言語による内部インターフェースの定義が必要となる。

我々は、データ項目をシーケンシャルデータの書式に加え、内部インターフェースについても「書式」ととらえ同様なスキーマで記述する方法をmappingSchema で用意した。この場合、変数や関数呼び出しといった **container** がデータの項目を保持することになる。

シーケンシャルデータの生成･解釈はこれら二つの書式の間のマッピングとして定義 (**mapping definition**)する。最も単純な **mapping definition** では、データの項目を、上流側、下流側で並び順を保ちつつ書式の変換をおこなう。

ここでは、例として、図2に示すような、シーケンシャルデータを受け取り、解釈し処理するプログラムを考える。**mapping definition** の記述に対応するのはこの図で紫色の部分である。mappingSchema の文法に従った **mapping definition** の例をリスト3に示す。また、これとインターフェースするプログラム(図2の紫色で示す部分の左右)の例をリスト4に、**mapping definition** の処理を行うプログラムをリスト5に示す。

mapping definition は、
- ー上流側のスキーマに固有な **schema element**
- ー上流･下流側のスキーマに共通な **schema element**
- ー下流側のスキーマに固有な **schema element**

の組み合わせで構成される。これらは3つの異なる **space** に属するものとして区別される。XMLを用いた mappingSchema の表記法では、**schema element** が属する **space** の違いを、XMLの要素が属するXML名前空間(ref 3)を用いて表現する。

ある **mapping definition** から、上流側の **space** と上流･下流に共通な **space** に属する **schema element** を抜き出すことで、上流側のスキーマを得ることができる。同様に、下流側の **space** と上流･下流に共通な **space** に属する **schema element** を抜き出すことで、下流側のスキーマを得ることができる。

シーケンシャルデータを用いた外部の仕様が先にきまる場合、シーケンシャルデータのスキーマを、まず用意し、内部のインターフェースで用いる **container** を追加していけば **mapping definition** を仕上げることができる。

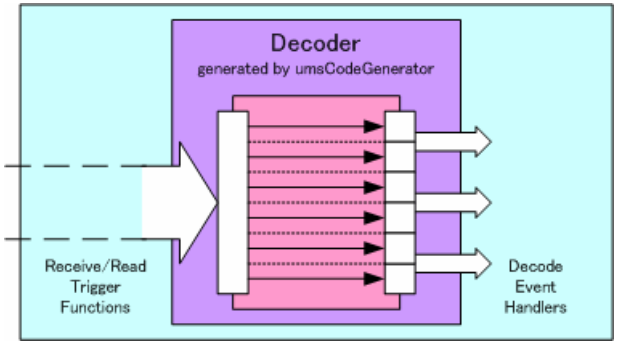

図2. シーケンシャルデータを処理するプログラムのイメージ

リスト3．**Mapping defintion** の例

```
<?xml version="1.0" encoding="UTF-8"?>
<perl:package name="txt2perl" xmlns:perl="http://ums.isas.jaxa.jp/0.3/perl/o"
        xmlns="http://ums.isas.jaxa.jp/0.3">
 <perl:decoderSub name="parse">
  <txt:byte xmlns:txt="http://ums.isas.jaxa.jp/0.3/dat/i" encode="txt">
   <group>
    <perl:readerSub name="userProgram">
     <group>
      <txt:byte length="6">
       <perl:var name="type"><data type="string"/></perl:var>
      </txt:byte>
      <txt:byte length="1">
       <txt:data type="string"><txt:param name="pattern">[¥s]+</txt:param></txt:data>
      </txt:byte>
      <perl:var name="name">
       <data type="string"><param name="pattern">[a-zA-Z_][a-zA-Z0-9_]*</param></data>
      </perl:var>
      <txt:value>(</txt:value>
      <txt:data type="string"><txt:param name="pattern">[¥s]*</txt:param></txt:data>
      <perl:var name="order">
       <data type="string"><param name="pattern">[0-9]+</param></data>
      </perl:var>
      <txt:data type="string"><txt:param name="pattern">[¥s]*</txt:param></txt:data>
      <txt:value>)</txt:value>
     </group>
     <zeroOrMore><loopGroup>
      <perl:array>
       <txt:data type="string"><txt:param name="pattern">[¥s]+</txt:param></txt:data>
       <perl:var name="value">
        <data type="string"><param name="pattern">[0-9]+</param></data>
       </perl:var>
      </perl:array>
     </loopGroup></zeroOrMore>
     <choice>
      <choiceGroup>
       <perl:var name="hasComment"><perl:value>1</perl:value></perl:var>
       <txt:data type="string"><txt:param name="pattern">[¥s]+</txt:param></txt:data>
       <txt:data type="string"><txt:param name="pattern">#</txt:param></txt:data>
       <perl:var name="comment">
        <data type="string"><param name="pattern">.+</param></data>
       </perl:var>
      </choiceGroup>
      <choiceGroup>
       <perl:var name="hasComment"><perl:value>0</perl:value></perl:var>
      </choiceGroup>
     </choice>
    </perl:readerSub>
   </group>
  txt:byte>
  erl:decoderSub>
</perl:package>
```

リスト4．シーケンシャルデータを解析するコードを呼び出すプログラムと解析されたデータ項目を処理するプログラムの

```
#!/usr/bin/perl

require("fixedSample.pl");

while(<>){
  s/(¥r)?¥n$//;
  &txt2perl::parse($_); # シーケンシャルデータの解釈ルーチンの呼び出し
}

# 解析されたデータ項目を処理するプログラム

sub txt2perl::userProgram {
   my $type          = $txt2perl::type;
   my $name          = $txt2perl::name;
   my $order         = $txt2perl::order;
   my @value         = @txt2perl::value;
   my $hasComment    = $txt2perl::hasComment;
   my $comment       = $txt2perl::comment;

  print( "$type $name($order) " );
  for(my $i=0; $i<$order; $i++){
    print( "$value[$i] " );
  }
    if( $hasComment ){
     print "#$comment";
    }
     print "¥n";
}
```

←

リスト3は、リスト1と比較し、赤で示す部分が加えられている。

Step 3 Mapping definition で定義される処理の自動生成

シーケンシャルデータの生成･解釈を行う部分とその他の部分のソフトウエアのインターフェースは、mapping definition で規定できる。これを満たすようなソースコードを手で書くことが可能である。一つの mapping definition を実現する方法には多様な実現方法がある。

Mapping definition で規定される処理の実現を、計算機上の処理系にゆだねることも考えられる。こうすれば、手作業のプログラミングを無くすことができるし、仕様に沿った厳密な処理が可能となる。

このような処理系には、幾つかのタイプが考えられる。Mapping definition に対応する専用な処理プログラムのソースコードを生成するものや、動作時に mapping definition を読み込む汎用なライブラリなどである。各々の処理系がどう mapping definition を実現するかは、手でプログラムを書いたプログラムと同様、多様でかまわない。扱う構造を制限すれば、これに特化した最適化もできる。

我々は、このようなプログラム作成が有効に機能することを実証するため、mapping definition から、シーケンシャルデータの解釈･生成を行う処理プログラムを自動生成する処理系の開発に着手した。これまでに、シーケンシャルデータを解釈しソフトウエアに渡すソースコードを生成するタイプの処理系のプロトタイプ (umsCodeGenerator) が動作している。リスト4は、実際に umsCodeGenerator が出力したものである。

← というのは、後付のストーリ。実際は、筆者らが作成している観測機器の状態を表示するソフトウエアにおいて、SIB (ISAS衛星の設計情報データベース)を参照する必要があったから。既存のSIB は書式が難解であるにも関わらず、実際の利用者は自前で SIB のテキストデータを解釈しなければならないという問題を抱えている。今回のツールを用いると、このような事例に比較的容易に対応できる。

umsCodeGenerator の 0.3版 では、シーケンシャルデータを入力し、3つの言語(Perl、C、JAVA) にデータ項目を渡す処理を生成できる。そのために、5つのURI

http://ums.isas.jaxa.jp/0.3/dat/i (シーケンシャルデータの入力)
http://ums.isas.jaxa.jp/0.3（入力から出力に渡されるもの)
http://ums.isas.jaxa.jp/0.3/perl/o (Perl言語 への出力)
http://ums.isas.jaxa.jp/0.3/clng/o (C言語への出力)
http://ums.isas.jaxa.jp/0.3/java/o (JAVA言語への出力)

で指定されるXML名前空間を用いている。

mappingSchema

まとめ

1. スキーマ言語を用いることで、一般的なシーケンシャルデータにおいても書式を厳密に記述することができる。その言語の (mappingSchema) ひとつを示した。

2. マッピング記法を用いてシーケンシャルデータをどうプログラムが受け取るか、内部のインターフェースとの対応を記述することで、プログラムのシーケンシャルデータを解釈する部分とその他の部分のインターフェースを厳密に規定することができる。その記法のひとつを示した。

3. マッピングの記法から、プログラムのうちシーケンシャルデータを解釈･生成する部分を処理系を用いて自動生成することで、厳密な処理を簡便に実現できる。このような処理系のプロトタイプ(umsCodeGenerator) を作成している。

まずは使ってみてください

http://ums.isas.jaxa.jp/

今後
- 必要に応じ、スキーマが表記可能な書式、処理系が処理可能な対象を広げていく
- 文法上の改善点が発見されればその改定を行う

Referrence

1. “Space Wire を用いた次世代衛星データリンクの開発”, Nomachi et al, The 5th Space Science Symposium, 2005
2. Document Schema Definition Languages (DSDL) -- PARTS2 Regular-grammar-based validation -- RELAX NG, ISO/IEC FDIS 19757-2
3. XML-Names, Namespaces in XML, W3C Recommendation, 14 January 1999, available at http://www.w3.org/TR/2000/REX-xml-20001006