Génération de décodeurs de formats binaires

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 janvier 2007. Dernière mise à jour le 12 juillet 2011

https://www.bortzmeyer.org/binaire-vers-structures.html

Le problème original a été défini par Bertrand Petit mais est apparemment « dans l'air » depuis un certain temps. Il s'agit de créer une bibliothèque de génération de décodeurs de formats binaires. L'idée serait de décrire le format dans un DSL, et un programme générerait un décodeur qui lirait le fichier binaire, contrôlerait les valeurs et remplirait des structures de données.

Le langage ASN.1 et ses outils associés ne conviennent pas, à cause du A dans ASN.1. Il veut dire "*Abstract*". Même si cet adjectif n'est pas tout à fait exact (par exemple, la définition d'un terme énuméré en ASN.1 nécessite d'indiquer concrètement les index), on ne peut pas utiliser ASN.1 pour décrire un format existant, un format que je n'ai pas choisi (prenons PNG ou libpcap <http://www.tcpdump.org/> comme exemple).

ASN.1 décrit le schéma de données. Il ne permet pas d'indiquer le placement des champs, le nombre de bits, etc. (Il existe toutefois une norme qui pourrait aider, il faudrait tester sa mise en œuvre en logiciel libre <https://github.com/ttsiodras/asn1scc>, ECN - "*Encoding Control Notation*" - ou ITU X.692 <http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.692-200811-I!!PDF-E&type=items>, suggérée par Jean-Marie Kubek.)

Voici, par exemple, dans un pseudo-Lisp, ce que pourrait être la description d'un format binaire :

```
(type GENDER (enumeration ('male.1 'female.2))) ; ISO 5218

(define 'file (sequence-of 'record))

(define 'record ('header 'body))

(define 'header ('version 'name (padding (+ (length name) self = 64)
        (if (>= version 4) 'gif-image) 'gender 'eoh))
; gif-image is defined in another module

(define 'version UNSIGNED_INT_32_BE)

(define 'eoh (value:0x0 size:32))
```

```
(define 'name (if (>= version 3)
                  (sequence-of UTF8_CHARACTERS)
                  (sequence-of ASCII_CHARACTERS)))

(name-of 'name completeName) ; Will be used in the produced decoder

(define 'gender GENDER)

(define 'body (sequence-of IEEE_754_FLOAT))
```

Je l'ai dit, l'idée est dans l'air : il existe plusieurs projets qui s'attaquent à ce problème. Le plus achevé est PADS <http://padsproj.org/> (merci à Bill Fenner pour l'idée), dont le logiciel est désormais sous une licence libre <http://padsproj.org/License.html>. Parmi les articles présentant PADS, notons :

— *"PADS : A Domain-Specific Language for Processing Ad Hoc Data"* <http://www.padsproj.org/papers/pldi.pdf>, une bonne synthèse de PADS,
— *"The Next 700 Data Description Languages"* <http://www.padsproj.org/papers/popl06.pdf>, qui généralise le problème et parle des alternatives à PADS.

Voici un exemple de code PADS :

```
Precord Pstruct summary_header_t {
   "0|";
   Puint32        tstamp;
};
Pstruct no_ramp_t {
   "no_ii";
   Puint64 id;
};
Punion dib_ramp_t {
   Pint64      ramp;
   no_ramp_t genRamp;
};
Pstruct order_header_t {
         Puint32              order_num;
 '|'; Puint32                 att_order_num;
 '|'; Puint32                 ord_version;
 '|'; Popt pn_t               service_tn;
 '|'; Popt pn_t               billing_tn;
 '|'; Popt pn_t               nlp_service_tn;
 '|'; Popt pn_t               nlp_billing_tn;
 '|'; Popt Pzip               zip_code;
 '|'; dib_ramp_t              ramp;
 '|'; Pstring(:'|':)          order_type;
 '|'; Puint32                 order_details;
 '|'; Pstring(:'|':)          unused;
 '|'; Pstring(:'|':)          stream;
 '|';
};
Pstruct event_t {
   Pstring(:'|':) state;     '|';
   Puint32         tstamp;
};
Parray eventSeq {
   event_t[] : Psep('|') && Pterm(Peor);
} Pwhere {
   Pforall (i Pin [0..length-2] :
             (elts[i].tstamp <= elts[i+1].tstamp));
};
Precord Pstruct entry_t {
   order_header_t header;
   eventSeq          events;
```

```
};
Parray entries_t {
   entry_t[];
};
Psource Pstruct out_sum{
   summary_header_t h;
   entries_t           es;
};
```

D'autres outils proches de PADS existent :

- La "*bit syntax*" <http://www.erlang.se/euc/99/binaries.ps> du langage Erlang,
- Construct <http://construct.wikispaces.com/>, un constructeur d'analyseurs en Python, qui fonctionne aussi bien pour analyser du texte que du binaire,
- binpac <http://www.imconf.net/imc-2006/papers/p29-pang.pdf>, "*A yacc for Writing Application Protocol Parsers*", un excellent outil orienté vers les protocoles réseaux, qui est livré en logiciel libre <http://bro-ids.org/wiki/index.php?title=BinPAC>,
- Le langage ROHC-FN décrit dans le RFC 4997[1],
- Le langage PacketTypes <http://www.sigcomm.org/sigcomm2000/conf/paper/sigcomm2000-9-2.ps.gz>, dont je ne trouve pas d'implémentation publiée,
- Le langage DataScript <http://datascript.sourceforge.net/>, qui semble avoir été implémenté depuis <http://datascript.berlios.de/>.
- L'outil Nail, décrit dans « "*Nail : A Practical Interface Generator for Data Formats*" <https://people.csail.mit.edu/nickolai/papers/bangert-nail-langsec.pdf> ». Le code est en ligne <https://github.com/jbangert/nail> sous une licence libre (avec un analyseur DNS comme exemple…).

Il existe d'autres outils, soit très sommaires soit assez éloignés du cahier des charges :

- La bibliothèque Hachoir <http://hachoir.org/>, pour les programmeurs Python, mais qui n'a pas de langage de description des fichiers binaires,
- Un exemple très intéressant <http://www.gigamonkeys.com/book/practical-parsing-binary-files.html>, où le DSL est embarqué, ici dans Common Lisp (il y a un joli exemple d'analyseur <http://www.gigamonkeys.com/book/practical-an-id3-parser.html>),
- Un gros projet ISO <http://metadata-standards.org/20944/>, qui semble une usine à gaz traditionnelle de cette organisation,
- Lumas, conçu pour décrire les en-têtes pour les protocoles réseaux, spécifié dans un "*Internet-Draft*", draft-cordell-lumas, mais qui n'a apparemment pas de mise en œuvre,
- TSN.1 <http://www.protomatics.com/tsn1.html>, un langage, et des logiciels <http://www.protomatics.com/products.html> (tous non libres),
- File-Spector <http://file-spector.sourceforge.net/> (très alpha, et uniquement pour la visualisation),
- <http://www.bitpim.org/pyxr/c/projects/bitpim/src/protogen.py.html>, très peu documenté,
- Un langage en XML <http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/idebug/help/en_US/debug390/tasks/tbcmapst.htm>, bien sûr,
- Le langage CSN.1 <http://www.dafocus.com/encodix_csn1.html> dont une spécification non-officielle <http://perso.orange.fr/cell.sys/csn1.htm> semble en ligne.

On le voit, il n'existe pas encore de solution idéale. Si un programmeur ambitieux lit ceci, je lui offre le cahier des charges :

- Concevoir le langage de description. Il doit avoir une bibliothèque de types de base (comme UNSIGNED_INT_32_BE) et la possibilité d'en créer. Il doit permettre de décrire des enregistrements composés de plusieurs champs (avec des cas où l'existence ou la taille du champ dépend de la valeur d'un autre champ). Les champs doivent pouvoir être nommés (cf. dernier point) Il doit permettre de spécifier le positionnement exact des champs.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4997.txt

— Le langage doit pouvoir prévoir le compactage (”*padding*”), que le membre de la structure aie un nom différent du champ, doit permettre de définir les noms des valeurs d'un champ correspondant a une enumération, doit permettre de définir des blocs qui pouront etre réutilisés dans les descriptions d'autres structures connexes, etc. Le langage doit évidemment avoir tout ce qu'on attend d'un langage moderne, comme la modularité (possibilité de définir un format à partir d'autres, par exemple MPEG audio d'un côté et MPEG vidéo de l'autre, puis utilisation simultanée dans une définition de MPEG tout court). Peut-être faut-il prévoir également un échappement vers du code arbitraire, pour traiter des cas compliqués comme les séquences de bits interdites dans les flux MPEG (je ne suis pas 100 % sûr car un tel échappement annule le côté ≪ sûr ≫ du langage de description).
— Écrire le programme qui va lire une description écrite dans le langage ci-dessus et la transformer en un ”.h” (ou équivalent pour votre langage favori) déclarant les structures et un ”.c” contenant les routines d'encodage et de décodage/vérification.