PACKEDOBJECTS

John P. T. Moore
Zedstar Dot Org Ltd
PO BOX 119, Pinner, HA5 2UR, United Kingdom
Email: jmoore@zedstar.org

Keywords: Packed Encoding Rules, ASN.1, Scheme, C

Abstract: *packedobjects* is a highly portable, cross platform, data encoding tool. The project is based on the telecommunications standard Packed Encoding Rules (PER). An abstract syntax language is used to define a protocol specification. *packedobjects* uses the Scheme programming language to represent the protocol specification within a symbolic expression (s-expression). Using an s-expression provides a more dynamic approach over the traditional method of parsing the specification and producing high level language code. The output of applying *packedobjects'* encoding rules to a protocol specification is a concisely encoded bit stream suitable for application domains such as network games development and mobile application development. The project has been released under the terms of the BSD license.

1 INTRODUCTION

Although the use of text-based protocols on the Internet is widespread, binary protocols have many diverse application domains. Examples include: security, mobile telephony, videoconferencing, aviation and transportation (Dubuisson, 2000). The demand for binary network protocols has risen due to performance requirements of domains such as online gaming and mobile application development. Distinct approaches to building a binary protocol exist. These include "encoding by hand", applying some transfer rules to an existing text-based protocol or using a specification language designed to produce binary encodings from the onset. Encoding by hand uses specific programming language features to design a data structure suitable for serialising for transmission across a network (Isensee, 2004). Although it is often possible to obtain performance gains by applying ad hoc bit packing routines, the disadvantages are likely to outweigh any such advantage. Disadvantages include the development and debugging time that might be required to build robust portable solutions. A more favourable approach might be to leverage existing skills in markup languages and apply some transformation to the markup before it is sent across a communication medium. This is the approach favoured by the Wireless Application Protocol (WAP) (Open Mobile Alliance, 2001). In addition, work on Fast Web Services (Sandoz et al., 2003) marks an attempt to improve the performance of web services by binary encoding data rather than sending textual XML representations. In general, tests have shown that binary encoding can be significantly faster than textual encoding (ASN.1 Consortium, 2002). The approach taken by the *packedobjects* project is based on the telecommunication standard Abstract Syntax Notation One (ASN.1) (ITU-T, 1988b). ASN.1 has been an international standard since 1984 and has produced efficient encoding techniques such as Packed Encoding Rules (PER) (ITU-T, 1998). Support for ASN.1 has been built into languages such as Erlang, and tools exist ranging from freeware solutions to professional toolkits.

Development with ASN.1 usually involves transforming the protocol specification into a high level language suitable to be compiled into an application (Larmouth, 1999). The high level language used is independent of ASN.1 and dependent on the tool vendor support. An advantage of using a compiler is that the specification language can be extremely powerful and expressive without fears of hindering runtime performance. The disadvantage, however, is that ASN.1 could be considered complex and also requires special handling to provide extensibility of protocols designed to produce tightly packed bit streams.

packedobjects takes a more dynamic stance by using an s-expression from the Scheme programming language to represent the protocol specification. This approach not only eliminates the lexical obstacles faced transforming ASN.1 into a high level language, but also provides a simple method of extensibility by forcing a separation between the protocol and the compiled application. In addition:

- Existing tools with built in support for editing Scheme can be used, such as Emacs.
- The Scheme language can be used to provide error handling and reporting.
- The use of quasi-quote can provide a mechanism for replicating user-defined abstract types in ASN.1.
- Other language features such as comments can easily be added to the specification.

The disadvantage of applying such a dynamic approach is the loss of runtime performance. However, by using a practical subset of ASN.1 the amount of processing that is required can be constrained.

2 ENCODING RULES

ASN.1 provides notation for defining abstract values which carry information. Applying encoding rules to an abstract syntax produces a series of bits ready for network transmission. Various encoding rules have been defined, including variants within encoding rules themselves (Mitra, 1994). A clear advantage exists when separating the way in which information is represented on a communications link from the abstract syntax used to describe the information. Depending on the nature of the application, different encoding rules may be selected without requiring any change to the protocol specification. It therefore may be possible to exploit advancements made in encoding techniques over the years.

```
-- Baseball Card Abstract Syntax (BCAS)
BCAS DEFINITIONS ::= BEGIN
   BBCard ::= SEQUENCE {
        name IA5String (SIZE (1..60)),
        team IA5String (SIZE (1..60)),
        age INTEGER (1..100),
        position IA5String (SIZE (1..60)),
        handedness ENUMERATED {
                   left-handed(0),
                   right-handed(1),
                   ambidextrous(2)},
        batting-average REAL
   }
   myCard BBCard ::= {
        name "Casey",
        team "Mudville Nine",
        age 32,
        position "left field",
        handedness ambidextrous,
        batting-average {
                  mantissa 250,
                  base 10,
                  exponent -3}
   }
END
```

The above code is an example of how ASN.1 can be used to describe a baseball score card. The specification is given with corresponding values. Although some notation is unique to the standard, it is fairly intuitive. Indeed, it can be useful as a means of simply communicating a protocol in a verbal or written sense between application developers. The language provides a set of atomic and composite types. The example provided is comprised of the atomic types 'IA5String', 'INTEGER', 'ENUMERATED' and 'REAL' together with the composite type 'SEQUENCE'. The composite types are a collection of one or more atomic and/or composite types. To transform this abstract syntax into a concise bit stream we can apply PER. A PER specification may use visible subtype constraints to optimise the encoding. In the example, constraints are placed on the size of the 'IA5String' and 'INTEGER' type. Subtyping is a powerful mechanism to restrict the set of values that may be allowed for a given type (ITU-T, 1988a). Typically, subtyping is used to restrict the allowable range of an integer type, provide a maximum and/or minimum size for the length of a string, and place bounds on the number of iterations that may occur in a 'SEQUENCE OF' or 'SET OF' type. The ability to customise data types produces efficient PER encodings. Not only are less bits sent across the communications link but also more optimised encoder/decoder implementations can be built to handle specific protocols.

3 PACKEDOBJECTS

packedobjects is an open source project implemented in Chicken Scheme (Winkelmann, 2006) and the C programming language. The software has been made available (Moore, 2006) under the BSD license. Currently the project uses Chicken as its host language, although it is possible to embed Chicken within C and therefore use C as the host language [1]. Scheme is used to provide all the high level data handling routines while C provides all the low level bit manipulation. The project is highly portable. Chicken has been ported to numerous platforms and the C code conforms to the ANSI standard.

[1]The term "host language" is used to indicate the language the developer would be using.

One of the motives of the project was to allow prototyping and development on embedded hardware. The user has the choice of working within the interpreter or compiling the code. Using the interpreter provides a simple method for designing and testing protocol specifications on embedded hardware without the need to cross compile. *packedobjects* has been successfully tested on a handheld device running embedded Linux.

packedobjects provides a simple API. The core functionality comprises of three routines to pack, unpack, and free data. Additional routines exist to read and write data from a file descriptor which facilitates communication across a network. As previously stated, a protocol is specified using an s-expression. For example, the following code illustrates how *packedobjects* would represent the baseball card depicted earlier in ASN.1:

```
(require-extension packedobjects)

(define bbcard
  '(bbcard sequence
           (name string (size 1 60))
           (team string (size 1 60))
           (age integer (range 1 100))
           (position string (size 1 60))
           (handedness enumerated
                       (left-handed
                        right-handed
                        ambidextrous))
           (batting-average sequence
                            (mantissa integer ())
                            (base enumerated (2 10))
                            (exponent integer ())))))

(define bbcard-values
  '(bbcard
    (name "Casey")
    (team "Mudville Nine")
    (age 32)
    (position "left field")
    (handedness ambidextrous)
    (batting-average
     (mantissa 250)
     (base 10)
     (exponent -3))))
```

3.1 Grammar

A difference exists between the way *packedobjects* and ASN.1 handle 'batting-average' in the baseball card specification. *packedobjects* does not directly support the 'REAL' type. This highlights how *packedobjects* uses a subset of the ASN.1 data types to represent other types. In this example the 'REAL' type is represented by a composite type containing three atomic types. The complete grammar for a data type in *packedobjects* can be described as follows:

```
datatype =  "(" datatype_name composite_type
| atomic_type ")" ;
composite_type =  "(" "sequence"
| "sequence-of"
| "choice"
| "set" datatype { datatype } ")" ;
atomic_type =  "(" string
| integer
| boolean
| enumerated ")" ;
integer =  "integer" "()"
| "(" "range" range ")" ;
string =  "string"
| "octet-string"
| "bit-string"
| "hex-string" "()"
| "(" "size" size ")" ;

datatype_name =  symbol ;
digit =  "0".."9" ;
signed_n =  [ "+" | "-" ] digit { digit } ;
unsigned_n =  digit { digit } ;
range =  signed_n | "min" signed_n | "max" ;
size =  unsigned_n | "min" unsigned_n | "max" ;
boolean =  "boolean" ;
enumerated =  "enumerated" "(" symbol { symbol } ")" ;
```

Where 'symbol' is any valid symbol as defined by the Scheme programming language.

3.2 Extended example

```
(define protocol
  '(random choice
           (query sequence
                  (num integer
                       (range 1 512))
                  (min integer
                       (range -1000000000 1000000000))
                  (max integer
                       (range -1000000000 1000000000)))
           (response sequence-of
                     (n integer
                        (range -1000000000 1000000000)))))
```

The above code represents a simple protocol used to request a sequence of random numbers from a server. The server obtains the numbers from the site *random.org* using HTTP. The client may request up to and including 512 random numbers in the range of -1,000,000,000 to 1,000,000,000. For example the following query requests 4 numbers:

```
(define query
  '(random
    (query
     (num 4)
     (min -1000000000)
     (max 1000000000)))
```

Supplying values to be encoded is straightforward, however, care is needed when using the 'sequence-of' type. Each iteration of a 'sequence-of' type must be delimited as it may contain multiple items. Square brackets are used to highlight the iteration of numbers in the random number protocol response as follows:

```
(define response
  '(random
    (response
     [(n -841852048)]
     [(n 350371729)]
     [(n -99891633)]
     [(n -76431948)])))
```

This response indicates that the 'sequence-of' contains just one item (an integer) and this sequence repeats four times.

Figure 1 shows a comparison of the number of bytes transfered at the application layer using HTTP and obtaining the numbers via the *packedobjects* protocol. [2] The results show the *packedobjects* version offers significant improvements in performance in terms of bytes transfered.

This example together with the previous baseball card example has illustrated the usage of all the data types available in *packedobjects* apart from the 'set' and 'boolean' type. Supplying values to the 'set' type is exactly the same as the 'sequence' type. Finally, using a 'boolean' type simply requires supplying the value 1 or 0 to indicate true or false respectively.

3.3 Extensibility

packedobjects allows the following types to be extended:

- constraints on integers and strings
- enumerations
- choices
- set items

To maintain compatibility between applications of different versions requires obtaining the latest protocol. This protocol could reside on a server from which the application bootstraps. Another benefit of this approach exists in domains such as gaming where it is important to stay one step ahead of users who are

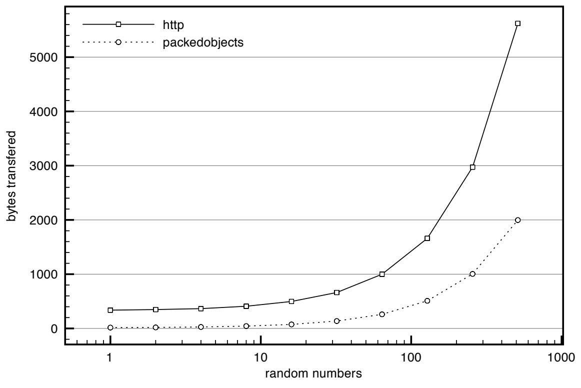

Figure 1: Total bytes transfered with HTTP and *packedobjects*

trying to reverse engineer the network protocol. Although using bit packed protocols provides some protection from casual observers, the dynamic approach of using s-expressions facilitates easy change to a protocol without requiring any patches to be applied to existing binaries. If the protocol is kept secure, then reverse engineering the bit stream is not straight forward.

3.4 Encoding deviations

Although the 'set' type is not often seen in PER specifications, it can provide a flexible addition to *packedobjects* protocols. The use of 'set' not only allows unordered items but also allows each item to be optional. This can provide support for the non extensible 'sequence' and 'sequence-of' types. In *packedobjects* the items of a 'set' are encoded in the order in which they appear in the specification. In PER order is defined by "the tag value" of the data type. The use of optional items requires a bitmap to be encoded to indicate which items are present. *packedobjects* encodes this bitmap as a 'bit-string'. This differs from PER which uses a constrained whole number approach. Using a 'bit-string' provides a simple way to support unlimited length sets and avoids the limitations described in the the following section. The disadvantage of using this approach and using sets in general is that they are less efficient in terms of the processing overhead required.

3.5 Limitations

The number of choices or enumerations is restricted to 2^{30} values. In practice this should be adequate, however, additional support could be provided by tailoring the specification accordingly and/or using the

[2] Due to the nature of random numbers the data between tests will vary, however, this will have little impact on the results displayed.

’set’ type. This limit also matches the limit imposed by the Chicken Scheme language for fixnums. Therefore, all integers are limited to fixnum ranges.

4 FUTURE WORK

A study of different protocol specifications could highlight which features of ASN.1 are most widely used. Often, however, protocol specifications developed in ASN.1 are not available in the public domain. The *packedobjects* project could benefit from such a study to help determine the most optimum subset of the language to support. A trade-off will exist between supporting a wide range of the language and runtime performance. When CPU performance is critical the *sfsexp* project could be examined (Sottile, 2005). The *sfsexp* project provides a library for working with s-expressions from C/C++. In addition, the specification language used by *packedobjects* is minimal and therefore it is possible to build optimised custom interpreters for other high level languages.

Whether or not CPU performance is an issue depends on the application domain. The cost of packing network messages may be negligible when the size of the network packets are small. In some networks, such as mobile networks, users are charged for the quantity of data they transfer. CPU performance again may not be deemed the main design factor, especially as the power of embedded hardware increases. Ultimately, it is the concept of throughput which determines overall performance of a network protocol. Throughput takes into account the cost in terms of CPU performance as well as the cost in terms of ”bits on the wire”.

5 CONCLUSION

The *packedobjects* project demonstrates the suitability of mixing the Scheme programming language and the C programming language to build a dynamic data encoding tool capable of producing concise binary network protocols. The increasing demand for binary protocols is being fueled by web developers, games developers and mobile application development. Along with this increasing demand is an increase in the number of hardware platforms to support. The range of high level languages available to do the same or similar job is also increasing. *packedobjects* uses an s-expression to describe a network protocol in a way that is independent of the programming language used or the target hardware platform.

The project has been designed to meet the demands of ubiquitous embedded computing and is ideally suited to bridge a gap between ad hoc bit packing and more formal approaches such as those offered by ASN.1 based solutions.

REFERENCES

ASN.1 Consortium (2002). Benchmark review : Comparison between binary encoder vs. textual encoder. http://www.asn1.org/benchmark/benchmark1.htm.

Dubuisson, O. (2000). *ASN.1 Communication between Heterogeneous Systems*. Morgan Kaufmann.

Isensee, P. (2004). *Bit Packing: A Network Compression Technique*, chapter 6, pages 571–578. Games Programming Gems 4. Charles River Media.

ITU-T (1988a). Abstract syntax notation one (ASN.1): Constraint specification. Rec. X.682.

ITU-T (1988b). Information technology – abstract syntax notation one (ASN.1): Specification of basic notation. Rec. X.680.

ITU-T (1998). ASN.1 encoding rules: Specification of packed encoding rules (PER). Rec. X.691.

Larmouth, J. (1999). *ASN.1 Complete*. AP Professional.

Mitra, N. (1994). Efficient encoding rules for ASN.1 based protocols. Technical report, AT&T Technical Journal.

Moore, J. (2006). packedobjects. http://www.call-with-current-continuation.org/eggs/packedobjects.html.

Open Mobile Alliance (2001). Wireless Application Protocol Architecture Specification. http://www.openmobilealliance.org/tech/affiliates/wap/wap-210-waparch-20010712-a.pdf.

Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M., and Pelegri-Llopart, E. (2003). Fast Web Services. http://java.sun.com/developer/technicalArticles/WebServices/fastWS/.

Sottile, M. (2005). the small, fast s-expression library. http://sexpr.sourceforge.net/.

Winkelmann, F. (2006). Chicken. http://www.call-with-current-continuation.org/.