Published in: IEEE INFOCOM '93 Proceedings, Volume 2.Los Alamitos: IEEE Computer Society Press, 1993, 647-653.

Towards tailoring protocols to application specific requirements

Philipp Hoschka

INRIA, Projet RODEO, 2004, Route des Lucioles B.P. 93, 06902 Sophia-Antipolis Cedex (France)
e-mail: hoschka@sophia.inria.fr

Abstract

A distributed application that uses a standard protocol for communication very often applies only a subset of the protocol options provided by the standard. However, it is difficult today for an application developer to tailor standard protocol options to the specific needs of an application, because no appropriate profiling method (formalized profile description and protocol configuration tool) exists. In this paper, we describe a prototype of such a method that is applicable to any protocol specified in ASN.1. We use the profiling of OSI application layer standards as a proof of concept. By applying our method to formalize the U.S. government profiles we were able to reduce the source code size of major parts of common OSI applications by 30 to 50%.

1. Introduction

Most of today's protocol standards contain so-called protocol options, i.e. communication services that will not be used by all applications. The technical reason for this is that a communication standard is usually intended to be used for different applications and different network technologies. Since the exact application/network combination cannot be foreseen at the time of the standard definition, optional protocol components are introduced to guarantee wide applicability. The recent advent of multimedia applications and high-speed networks potentially increases the number of network/application combinations and, thus, the requirement for options in protocol standards. A less technical, but nevertheless important reason of protocol options is the "design by committee" method used for most protocol standards: here, social processes often cause the introduction of several protocol options with similar functionality, because the committee could not come to an agreement on the competing solutions.

A protocol standard with many options can pose serious problems for practical use of a protocol suite, since the implementation of the options may result in protocol implementations that are slow and have a large code size. These problems of general-purpose standard protocols resulted in the development of customized protocols for certain application classes (e.g. [Yeon92], [Rose88]). However, the customized protocol are incompatible with standard protocols. Moreover, it is a major effort to design, verify and implement a new protocol. Since a customized protocol has fewer options, it is also less flexible and usually cannot be easily adapted to other application/network combination than the one it was designed for. Despite these drawbacks, customized protocols demonstrate what can be gained in terms of execution speed and code size by tailoring a protocol to application specific requirements. Experiments reported in [Rose90] comparing applications built on top of the lightweight presentation protocol with applications built on top of a library of general OSI protocols (ISODE) have shown that the former are on average 100% faster and 50% smaller than their ISODE counterparts.

To reduce code size and increase execution speed of general purpose protocols without the disadvantages of customized protocols a *profiling method* is required. Such a method consists of two parts: first, a formalized *profiling language* that allows the definition of the set of protocol options used by a certain application/network combination (the *profile*); and second a *protocol configuration tool* that adapts a formalized specification of the standard protocol to this profile. From this adapted specification, one can configure or generate the code necessary to implement the selected options of the general purpose protocol.

In this paper, we describe a prototype of a profiling method that facilitates protocol configuration for application programmers. The profiling language is based on ASN.1 (Abstract Syntax Notation one ([CCIT88], [Stee90])). ASN.1 is a type description language widely used to specify the structure of messages exchanged by OSI protocol. These messages are referred to as Protocol Data Units or PDUs. ASN.1 is also used outside the realm of OSI, e.g. to specify the Kerberos and WAIS protocols. The protocol configuration tool for the profiling

method is still under development, and hence the ASN.1 specifications of standard protocols are adapted manually to fit the requirements of the profile.

In section 2, we further motivate the need for a formal description of an application specific profile by analyzing the effect of protocol options on the code size of OSI implementations, and discuss several approaches to code size reduction. Section 3 presents an ASN.1-based profiling method and demonstrate its use by an example. In Section 4 we present measurements of the code size reduction achieved by the application of our profiling method to an OSI protocol implementation. In section 5, we conclude with a critical analysis of our preliminary profiling method and some ideas for future applications and extensions.

2. Effect of protocol options on code size

2.1. A case study in standard protocol implementation

The RODEO group of INRIA implemented a prototype of a software tool to implement applications that use the OSI ROS (Remote Operation Service) protocol (*Basicwriter*, [Bust92]). The ROS protocol implements a service that follows the remote procedure call paradigm and is used in OSI applications like X.400 (electronic mail), X.500 (directory service) and CMIP (network management). ROS applications are good examples of applications that use only a subset of the functionality defined in a standard protocol. The profile for ROS applications (without the use of reliable transfer service element) requires only two of the twelve functional units defined in the OSI session service standard, and one of the three functional units defined in the OSI presentation service standard.

Basicwriter uses a library that implements the routines of the presentation/session layer required by ROS. The library is implemented in C and consists of two parts: a hand-coded, integrated protocol machine for the session/ presentation protocol and the encoding/decoding routines that convert between the local and the external presentation of a PDU (also called *marshalling*). The encoding/decoding routines were automatically generated using the MAVROS ASN.1 compiler [Huit91].

In building a simple example application using Basicwriter, we noticed that their code size seemed larger than necessary. In fact, application of the profiling method presented later in this paper reduced the binary code size of this application by 22%. Using the UNIX profiling tool, we discovered a high number of unused encoding/decoding functions for PDUs and PDU-fields originating from unused protocol options of the presentation/session layer. These routines had been linked into the application code since the ASN.1 specifications used to build the Basicwriter library were taken directly from the standard in the case of the presentation protocol, and directly reflected the standard in the case of the session protocol (the session protocol is not specified in ASN.1 in the standard). Thus, the ASN.1 texts contained specifications for all options of these standards, including the options unused by ROS applications.

2.2 Possible approaches to code size reduction

A large code size of a protocol implementation puts extra load on system resources (e.g. main memory, hard disks). It is especially critical if the software is to be run on machines with tight memory restrictions, e.g. mobile systems. Thus, we investigated methods that would allow us to eliminate the superfluous code from applications built with Basicwriter.

Since in our case the unused code in OSI applications is caused by the use of automatic code generation by an ASN.1 compiler, one possible solution is to use *hand-coded encoding/decoding routines* specifically tailored to the application profile. However this increases the development time considerably.

Another thought that immediately comes to mind when an application contains too much superfluous code is that this code should not have been linked into the application in the first place. In our case, this is caused by the API (Application Programer Interface) to the encoding/decoding routines generated by the ASN.1 compiler. However, we did not want to implement a *modified API*, not only for practical reasons - a lot of existing code depends on it - but also for technical reasons. It turns out to be very difficult to design an API for C encoding/decoding routines that considerably reduces the amount of unused code linked to an application and is still reasonably practical to use. While the use of object-oriented programing languages for encoding/decoding routines might help to alleviate this problem, a major advantage of ASN.1 would be lost by this: namely the use of ASN.1 for mixed language programming (the RODEO group implemented a version of MAVROS that generates Lisp code, a C++ version is planned). Thus, a technique to reduce the application code size should be independent of features of the particular language used to implement the encoding/decoding routines.

Encoding/decoding by an interpreter is another approach to reduce the code size of ASN.1 based applications (e.g. used by PEPSY in ISODE [Onio89]). However, interpreters considerably slow down protocol processing. This becomes more and more unacceptable today, where the availability of high speed network technology makes the overhead of protocol processing in end-systems the bottleneck in a communication system. Thus, when the code size of encoding/decoding routines is a problem a first step is to tailor the standard ASN.1 specification to the application profile. Only if after this step the code is still too big, encoding/decoding by an interpreter should be considered.

One method that can be used to tailor the ASN.1 specification to the application profile is *manual editing* of the ASN.1 specification. While this approach might be acceptable as a 'quick-and-dirty' solution, it has two major disadvantages. First, it is error-prone, since one can not verify automatically that the resulting tailored specification still satisfies the configuration constraints imposed by the standard. For instance, there is no guarantee that one did not accidentally delete a mandatory PDU field, and that consequently the tailored specification is incompatible with the standard. Second, standard ASN.1 specifications usually consist of several pages, so that keeping track of the modifications made to the standard specifications quickly becomes difficult when the number of profiles grows. Moreover, it is difficult to decide from manually adapted specifications whether two implementations of an application use the same profile, or in which respect their implementation has to be changed to make them interoperable.

A better approach is to have a profile document that describes the application profile in a concise, formalized way. A *protocol configuration tool* can take this document and the standard ASN.1 specification as input. The tool verifies that the profile satisfies the constraints imposed by the standard and automatically rewrites the standard ASN.1 specification to conform to the profile document. In the following section we demonstrate how such an application profile document can be written in standard ASN.1.

3. A profiling method based on ASN.1

3.1 General approach

The ASN.1-based profiling method starts from two documents: first, an ASN.1 module that defines the structure of a protocol's PDUs including all optional elements; and second, an informal description of the application profile that describes which of the protocol's optional services are actually used by the application, and which are not.

The optional services mentioned in the informal profile specification have to be translated into concepts that are meaningful in ASN.1 The used and unused optional services map onto one of the following cases:

- *Unused PDU*: one of the PDUs that is specified in the standard will never be used by the application.
- *Unused optional PDU field*: a PDU field that is defined as optional in the standard will never be used by the application.
- *Used optional PDU field*: a PDU field that is defined as optional in the standard will always be used by the application.

These cases can be expressed elegantly using the *ASN.1 subtyping facility*. Instead of giving a general introduction into ASN.1 and its subtyping facility, we demonstrate the use of the profiling method by an example in the following sections.

3.2 Example ASN.1 specification: a protocol with options

Since examples of real ASN.1 specifications found in protocol standards are too complex to present here, we wrote the example ASN.1 specification shown in Fig. 1 It contains elements of the OSI session and presentation PDUs.

The specification starts with the definition of the set of PDUs that are used by our example protocol by an ASN.1 `CHOICE` construct. The protocol provides services to build up and tear down a connection (`connect`-, `accept`-, `refuse`- and `disconnect`-PDUs). Moreover, the protocol offers two alternative services for transmitting application data: expedited data transfer (`expedited-data`, `expedited-data-ack`-PDUs) and normal data transfer (`data`-, `data-ack`-PDUs).

The next section of the specification describes the structure of each of the single PDUs of the protocol in turn. When requesting a connection, the caller may transmit its address in the connect-PDU, e.g. because the caller requires simple authentication. Additionally, the `user-data` field might be used to transmit an encrypted password. The specification of the other PDUs has been left out to keep the size of the example small.

```
generic-protocol DEFINITIONS::=
BEGIN
-- Definition of the set of PDUs
Generic-PDUs::= CHOICE {
   connect       [0]       Connect,
   accept        [1]       Accept,
   refuse        [2]       Refuse,
   disconnect    [3]       Disconnect,
   data          [4]       Data,
   data-ack      [5]       Data-ack,
   expedited-data[6]       Expedited-
                           data,
   expedited-data-ack[7]   Expedited-
                           data-ack
}
-- Definition of the Connect PDU-type
Connect::= SEQUENCE {
   calling-address         OCTET STRING
                           OPTIONAL,
   called-address          OCTET STRING,
   user-data               OCTET STRING
                           OPTIONAL
}

-- Definition of the other PDUs left
   out
-- Accept::= ...

END
```

FIGURE 1. Example ASN.1 specification

```
tailored-protocol DEFINITIONS::=
BEGIN
IMPORTS Generic-PDUs, Connect
   FROM generic-protocol;

Tailored-PDUs::= Generic-PDUs
   (WITH COMPONENTS{...,
        connect (INCLUDES
                  Tailored-Connect),
        expedited-data          ABSENT,
        expedited-data-ack      ABSENT})

Tailored-Connect::= Connect
   (WITH COMPONENTS {...,
        calling-address         PRESENT,
        user-data               ABSENT})
END
```

FIGURE 2. Tailoring ASN.1 specification of Fig.1 to application-specific requirements

3.3 Example use of profiling method: a tailored protocol

Consider now an example application that will use the generic example protocol, but with the following profile:

- *Unused optional PDUs*: The application does not require the expedited data transfer service. Thus, the PDUs `expedited-data` and `expedited-data-ack` are never used.
- *Unused optional PDU fields*: On connection set-up, the application does not need to send any application specific data such as a password. Thus, the `user-data` field of the `connect`-PDU is never used.
- *Used optional PDU fields*: On connection set-up, the caller has to provide its address for authentication purposes. Thus, the optional `calling-address` field is always present.

Figure 2 shows how these requirements are expressed in a profile document. First, the profiling module imports all the symbols that it will use from the profiled module, i.e. type references of the PDUs and fields that will be constrained. Then, the particular restrictions of the application are defined:

- *Unused optional PDUs*: The unused PDUs are eliminated by using the *inner subtyping* construct of ASN.1. With this construct, one can specify that the `CHOICE` defining the permissible PDUs for the protocol will not contain the two alternatives `expedited-data` and `expedited-data-ack` by defining them as `ABSENT`. Furthermore, in the `Tailored-PDUs` subtype, the *contained subtype* construct of ASN.1 is used to indicate that the `connect` PDU is further constrained by a subtype specification named `Tailored-Connect`.
- *Unused optional PDU fields*: The PDU field `user-data` is eliminated in the `Tailored-Connect` subtype definition by defining it as `ABSENT`.
- *Used optional PDU fields*: the calling-address field is set to `PRESENT`.

Figure 3 shows a complete and exhaustive definition of the Profiling Method. It relates all possible cases of ASN.1 type constructs T that can occur in a standard protocol specification to the restrictions that may occur in a profile, and shows the appropriate subtype specification for each case. We dropped the distinction between unused

Restriction on type T	CHOICE, SEQUENCE, SET	SET OF, SEQUENCE OF
Optional Component C unused/used	`Tailored-T::= T (WITH COMPONENTS {..., C ABSENT/PRESENT} )`	(not applicable)
Component C restricted by subtype R	`Tailored-T::= T (WITH COMPONENTS {..., C (INCLUDES R)})`	`Tailored-T::= T (WITH COMPONENT (INCLUDES R))`
No restriction	`T::= T`	`T::= T`

FIGURE 3. Profiling method: Summary

Profile	LOC encoding/ decoding	Savings by profiling
Full ASN.1 specification	6069	-
X.500, FTAM, CMIP, MMS	3129	48%
X.400-88-P1	4091	32%
Virtual terminal protocol	4044	33%

FIGURE 4. Application of profiling method to GOSIP profiles

optional PDUs and unused and used optional PDU fields made in the example, since an ASN.1 specification does not distinguish between types that are PDUs and types that are PDU fields. The special case "no restriction" was introduced because it was required to describe profiles for some ASN.1-specified protocols found in practice (e.g. the OSI presentation protocol))

3.4. Automating the profiling method

In order realize the full advantages of a profiling method the formalized profile must be processed automatically by a software tool that serves as *protocol configuration tool*. For the ASN.1 based profiling method, two implementation approaches for such a tool are possible.

First, an ASN.1 compiler could directly generate encoding/decoding routines that handle only the subtypes contained in the profile. While this might seem an obvious approach, most existing ASN.1 compilers (e.g. Pepsy, MAVROS) do not take advantage of the subtyping information, mostly because the subtyping notation, and in particular inner subtyping, has seldom been used in protocol standards up to now.

A second approach is to implement a preprocessor that merges the original ASN.1 and the profile specification. In the resulting new ASN.1 specification, all type definitions and references from the original module that are defined as ABSENT in the profile do not occur, and the OPTIONAL definitions from the original specification that are defined as PRESENT are removed. The preprocessor approach has the additional advantage that its output is legal ASN.1 and can be processed by any existing ASN.1 compiler (e.g. [Onio89], [Neuf90], [Harv91], [Koiv92]). Thus, the optimisations achieved by profiling can be exploited independently of the ASN.1 compiler used.

4. Applying the profiling method to OSI standard protocols

To validate the practical usability of the Profiling Method, we used it to formalize the GOSIP profiles for OSI applications [NIST92]. Because an automatic preprocessor for profile documents was not yet available, we manually adapted the general ASN.1 specifications for the presentation and session layers to the requirements of the profiles. Then, we compared the lines of C code of the encoding/decoding routines generated by MAVROS before and after tailoring. Figure 4 shows the results for all three different classes of presentation/session-profiles that occur in the GOSIP profiles. The net savings vary with the number of unused protocol options. However, in all cases major reduction in source code size (between 32 and 48%) are achieved. Those will result in a reduction of overall application code size when the routines are compiled and linked to a specific application.

To study the effects on application code size, we applied the profiling method to the simple Basicwriter example application we analyzed in Section 2. After recompiling and relinking the application, the application binary code

was 22% smaller (reduction of 122 KB from 416 KB to 294 KB). Further reductions are possible, since our hand-coded protocol machine was not as easily adaptable to the application specific requirements as the ASN.1 specification. Real-world applications with more functionality like X.500 will benefit from this code size reduction, since ASN.1 coding/decoding routines usually take up a major portion of the application's code size when an untailored ASN.1 specification is used.

5. Discussion

The experiments with the prototype profiling method demonstrate that large reductions in application code size can be achieved.The size of these reductions depends on the amount of optional components contained in the standard protocol, which is high in the OSI protocol suite we used as an example. However, the problem of profiling (i.e. adaptation of protocols to application requirements) transcends the specific problems in the OSI suite, as the ongoing discussion about the quality of service of transport protocols shows. It will be interesting to see whether the principle of the profiling method (selection of optional functions by restricting PDU syntax) can also be usefully applied in this area.

In addition to code size reduction, it seems clear that the elimination of optional protocol data units and fields will also lead to speed advantages, since the overhead of checking for the presence of optional components is high in a protocol with many options. We will verify this with further measurements.

General techniques to reduce the size and increase the speed of the code generated by an ASN.1 compiler have already been studied elsewhere ([Huit92], [Onio89]). However, our ASN.1 based profiling method is the first to achieve these goals by exploiting application specific requirements - a protocol design paradigm that was proposed in [Clar90].

The ASN.1 subtyping mechanism we have chosen to realize the profiling method does not offer all desirable features of a such a method. For instance, it does not use the service abstraction of layered protocol suites. Instead, the author of a profile has to know how optional services are mapped onto the PDU-fields and PDUs in the standard ASN.1 definition. Experience will show whether a higher-level, service-oriented profiling method is required. Moreover, the ASN.1 specifications used in today's standards do not contain all configuration constraints of the protocol. For example, one cannot specify that a certain optional PDU is only present in the protocol if another one is also present. Consequently, the configuration tool cannot check in all cases whether a profile obeys the rules of the protocol standard. Extensions to standard ASN.1 are required before such verifications are possible.

6. Conclusion

Protocol options are required in a standard protocol suite so that the protocols defined can be used for a wide range of applications and networks. Moreover, protocol options seem to be an unavoidable by-product of the social process of protocol standard design. In both cases, software engineering solutions such as the prototype profiling method introduced in this paper are required to make the standard protocol suite useful to application programmers.

The profiling method allows to formally describe an application profile and to automatically adjust the protocol implementation to this profile. Applying a profiling method to a standard protocol may result in a considerable code size reduction for protocols with many optional components. We demonstrated that savings of about 22% of binary code size and between 30 to 50% of encoding/decoding routine source code size can be achieved for the OSI standard protocol suite.

7. References

[Bust92] Bustos, Anne-Marie. "Un environment OSI autour du compilateur ASN.1 MAVROS (An OSI environment built around the ASN.1 compiler MAVROS)." In French. Ph. D. Thesis. University of Nice Sophia-Antipolis, 1992.

[Clar90] Clark, David and David Tennenhouse. "Architectural Considerations for a New Generation of Protocols." SIGCOMM '90, 200-208, 1990.

[CCIT88] CCITT. Data Communication Networks - Open Systems Interconnection (OSI) - X.208. Specification of Abstract Syntax Notation One (ASN.1). Blue Book, Vol. 8. Melbourne, 1988

[Harv91] Harvey, James and Alfred Weaver. "Experience with the Abstract Syntax Notation One and the Basic Encoding Rules." Proceedings 16th Conference on Local Computer Networks. Los Alamitos: IEEE Computer Society Press, 1991.

[Huit91] Huitema, Christian. "MAVROS: Highlights on an ASN.1 compiler." Internal working paper. Sophia-Antipolis: INRIA, Project RODEO, 1991

[Huit92] Huitema, Christian and Ghislain Chave. "Measuring the Performance of an ASN.1 Compiler." Upper Layer Protocols, Architectures and Applications, Proceedings IFIP WG 6.5 Conference. Vancouver: 1992, 99-112.

[Koiv92] Koivisto, Juha and James Reilly. "Generating Object-Oriented Telecommunications Software using ASN.1 Descriptions." Upper Layer Protocols, Architectures and Applications, Proceedings IFIP WG

6.5 Conference. Vancouver: 1992, 127-142.

[Neuf90] Neufeld, Gerald and Y. Yang. “The Design and Implementation of an ASN.1 C Compiler.” IEEE Transactions on Software Engineering, 16, 10 (October 1990).

[NIST92] NIST. Stable Implementation Agreements of Open Systems Environment Implementors’ Workshop - Part 5: Upper Layers. March 1992.

[Onio89] Onions, Julian and Marshall Rose. “The Applications Cookbook.” Stefferud, Einar, Ed. Proceedings of the Fourth International Symposium on Computer Message Systems. Amsterdam: North-Holland, 1989, 217-231.

[Rose88] Rose, Marshall. ISO Presentation Services on top of TCP/IP-based internets. Request for Comments 1085, DDN Network Information Center, SRI International, 1988.

[Rose90] Rose, Marshall. The Open Book - A Practical Perspective on Open Systems Interconnection. Prentice-Hall. Englewood Cliffs,1990.

[Stee90] Steedman, Douglas. Abstract Syntax Notation One (ASN.1) - The tutorial and reference. Twickenham: Technology Appraisals, 1990.

[Yeon92] Yeong, Wengyik, Tim Howes and Steve Hardcastle-Kille. "Lightweight directory access protocol". Internet-Draft. DDN Network Information Center, SRI International, August 1992.