Increasing the Efficiency of Computer-aided Protocol Implementations

Thomas Held

Otto-von-Guericke-Universität Magdeburg
PF 4120, 39016 Magdeburg, Germany
fax: +49-391-5592-2810
e-mail: theld@irb.cs.tu-magdeburg.de

Hartmut Koenig

Technische Universität Cottbus
PF 101344, 03013 Cottbus, Germany
fax: +49-355-69-2236
e-mail: koenig@tucs1.rz.tu-cottbus.de

Abstract: *The paper discusses ways for increasing the efficiency of protocol implementations which are automatically derived from formal descriptions. The subject of our discussion is the implementation model used in the implementation tool for the transformation process. The implementation model defines the architecture of the final implementation and has a crucial influence on its efficiency. Using an experimental Estelle-C compiler we discuss the impact of design decisions for the implementation model on the efficiency of the implementation. As a consequence, the use of a variable implementation model is proposed.*

Keyword Codes: C.2.2; D.2.1; D.2.2

Keywords: Network Protocols, Requirements/Specifications, Tools and Techniques

1. Introduction

The performance of communication systems is decisively determined by the protocols used. The experience of practical protocol implementations shows that protocol performance depends as much, and usually more, on the implementation than on the design [Wats 87]. The development to high-speed communication has increased the impact of the implementation phase in the protocol life cycle. Besides the design of new streamlined protocol architectures, the application of new implementation technologies, such as parallel implementation techniques, is one of the main directions to support this development.

Nowadays, communication protocols are usually implemented by hand based on informal (verbal) specifications provided by ISO, CCITT or other organizations. This process is time-consuming, tedious and prone to error. Computer-aided protocol implementation¹ can increase the quality of the implementation [Agga 85], [Koen 90]. Although approaches for a computer-aided derivation of implementations have been investigated for a long time their practical application is limited. There are several reasons for this. First the technology of this approach is not yet fully developed to ensure an efficiency of the generated code which can be compared with the hand-coded one. Second, most of the systems reported in literature, aim for executing the FDTs by implementing their semantics rather than for product implementations. This kind of implementation serves first of all for validation purposes. The demands of a real implementation cannot be satisfied. Third, the implementations are mostly confined to one layer.

The implementation has attracted much less attention in the FDT community than the other phases of the protocol life cycle; this is because the implementation process itself provides few opportunities for formalization. On the other hand, the objective of the protocol life cycle is to produce efficient and correct implementations, the quality of which is greatly influenced by the implementation phase itself.

¹The notion *computer-aided protocol implementation* is supposed to condense the different notions found in literature such as automated implementation, semiautomated implementation or derivation.

The paper discusses ways for increasing the efficiency of computer-aided protocol implementations. The subject of our discussion is the implementation model of the derivation tool that determines the architecture of the implementation. Using an experimental Estelle-C transformer we discuss different aspects of the influence of the implementation model on the efficiency of an implementation.

2. Computer-aided implementation

The general principle of the computer-aided implementation is well-known and described in literature [Agga 85], [Chan 93]. Regarding existing tools carefully one can notice that the notion of implementation is applied in a broad meaning. To distinguish the different meanings we introduce the notion language and product implementation of a protocol (see Fig. 1).

Fig. 1: Forms of protocol implementations

A **language implementation** aims at the execution of the specification by implementing the semantics of the FDT. The derivation process can be fully automatized in the tool. This kind of implementation serves the purpose of validation. The demands of real protocol realization cannot be met. Most of the existing FDT-compilers aim at language implementations.

A **product implementation** is a realization of the protocol that is optimized for a concrete target environment. Compared with a language implementation, a product implementation must fulfill the following requirements:

- optimized adaptation to the target environment (hardware resources, operating system, user requirements),
- good performance parameters that meet real time constraints by introducing performance increasing strategies which are not necessarily contained in the formal description (but which do not hurt the semantics of the specification).

Current tools for computer-aided protocol implementation are scarcely able to meet the requirements of product implementations (in addition, it is seldom their objective). A practical way to use these tools for product implementations is the semi-automated implementation by using those parts of the generated code for the product implementation which fulfill the needed requirements.

To reflect the different approaches to implementation discussed above we introduce the following terminology for our discussion in accordance with the terminology used in software engineering [Nagl 90]. All tools which aim at a language implementation are called **compilers** and tools at a product implementation **transformators**.

3. Role of the implementation model

For computer-aided implementation, the semantic model of the FDT is mapped on the model of the implementation. The **implementation model** describes the logical structure of the implementation, its components and the principles of their interaction. It determines the architecture of the implementation. The implementation model chosen determines the transformation rules

for the derivation procedure. Its design is therefore of crucial importance for the whole implementation process.

Unlike a manual implementation where the implementation model is worked out during the implementation design phase, the design of the implementation model for a computer-aided implementation is part of the tool development, i.e., the implementation model is designed before and independently of the actual implementation. A further characteristic of an automatically derived implementation is that it prescribes a certain implementation model, i.e. the process and module structure, and the partition of the functions are the same in all implementations derived by this tool. The rigid application of the implementation model restricts the efficiency of the implementation because the transformation tool maps the abstract description on executable code portions applying firm transformation rules. It is not capable of considering local implementation matters or of giving the opportunity to introduce individual optimizations related to the implementation context. This can only be done by hand with great effort.

The way the implementation model is derived from the semantic model of the FDT is not prescribed as long as the semantics of the specification is preserved. For a compiler, the implementation model is defined by the FDT semantics. The transformation can be almost entirely automated. The implementation model of a transformer, however, is not bound to the semantics of the FDT to introduce implementation solutions which satisfy the efficiency requirements. But experience shows that the implementation model of transformers usually leans on the FDT semantics because this essentially simplifies the transformation process. On the other hand, it confines the utilization of different implementation strategies which can increase the performance by exploiting specific properties of the protocol. The endeavour to get as possible flexible and efficient transformations of the specification is in part limited as for certain features of the FDT it is difficult or even impossible to find automated applicable transformation rules. So it is, for instance, very difficult to map an Estelle specification onto the activity-thread model [Svob 89] that is known to be more efficient than the server model. Therefore, most of the implementation models for computer-aided implementation are based on simple principles (server model, asynchronous interfaces, only one layer).

The implementation model has, consequently, a decisive influence on the quality of a computer-aided implementation. Its complexity and completeness determine the possible degree of automation. To ensure that computer-aided implementation techniques, in particular transformers, are an applicable choice in practice it is therefore necessary to design implementation models that guarantee an efficient implementation and find a broad acceptance among users.

4. Evaluation of the implementation model

In this section, we discuss the influence of certain implementation decisions on the efficiency of the implementation. For this, we consider different parts of the implementation model of the Experimental Estelle-C Transformer (EECT) [FoHe 92], [Held 94]. The implementation model of the EECT used first was leaned on the NBS prototype compiler [NBS 87]. It has changed step by step. The EECT has been applied to different protocols. In this paper, we use the LLC-3 protocol [ISO 88] and an experimental protocol [Held 94] as reference protocols. The hardware basis for all measurements has been an Intel 80486 DX processor with 33 MHz frequency. As a measurement system, a software monitor has been used [Prof 91]. The number of events in the experiments was between 1000 and 10000 to get statistically representative results.

4.1 Process model

The mapping of the implementation model on the process structure of the target computer system constitutes a central issue of the protocol implementation. The process model repre-

sents the physical realization of the implementation model. There are mainly two approaches for the process model [Svob 89]: the server model and the activity-thread model.

The *server model* implements a protocol instance by a single process or by multiple processes. They communicate via messages. The *activity-thread model* is mapped on a set of procedures. The communication is realized by procedure calls. A process in this model is essentially a thread of procedure executions.

For computer-aided protocol implementation the server model is preferred due to its clear structuring principles. All Estelle compilers use this process model. It is also applied in our experimental compiler. Therefore, we consider only the server model in the following.

4.2 Scheduler

The scheduler is the "heart" of the implementation which realizes the Estelle semantics [ISO 89]. It is coded by hand and belongs to the implementation environment (see Section 4.4). In this section, we discuss several strategies to implement the scheduler. Our discussion is divided in two parts: selection of the fireable transitions and access to the program code.

4.2.1 Selection of the fireable transitions

We have implemented three approaches of the Estelle algorithm to determine the fireable transitions. They are compared in Figure 2 for the experimental protocol [Held 94]. Approach I is a one-to-one implementation of the selection algorithm as described in the Estelle standard [ISO 89]. This approach is used, for example, in [NBS 87]. Approach II is an optimized implementation of approach I. It keeps the Estelle semantics but it has a different algorithmic structure. The algorithm is described in [Held 94]. Both approaches consist of two step: (1) the fireable transitions are determined and (2) the fireable transitions are executed. The fireable transitions have to be buffered between steps (1) and (2).

Fig. 2: Share of the selection routines on the runtime

In approach III, the selection algorithm has changed in such a manner that the buffering of the transitions can be avoided. The changed algorithm differs from approach II by determining the set of fireable transitions. The modules are in turn searched for fireable transitions. The found transitions are fired immediately. The only difference in the effect of the algorithm is that the result of each transition (new interactions, change of the specification structure) has to be immediately taken into account.

This change has decreased the percentage of the selection function on the runtime of the total system to 39 percent (see Figure 2). The disadvantage of this approach lies in the fact that the order of the execution of the transitions can deviate from the original Estelle mechanism. This approach can be only applied if this deviation does not influence the conformance to the (verbal) protocol standard. This must be guaranteed by the implementor. The reason for this possible deviation from the original verbal protocol standard is caused by the semantics of

Estelle that introduces a fixed procedure for selecting transitions regardless whether it is needed in the concrete application.

4.2.2 Access to the program code

There are two possibilities to implement the access to the program code that belongs to the selected transitions: programmed and table-controlled access.

The *programmed access* realizes the access procedure by means of IF- or CASE-statements that can be directly derived from the specification of the transitions in Estelle. In the EECT, the derived statements are united in a transition procedure that is generated for each module. The *when*-, *from*- and *provided* clause are mapped on nested IF-statements according to the order of their appearance. The THEN-part of the inner IF-statements contains the transition blocks. For selection approach I and II, the transition blocks are stored separately. In this case, a call of a function is generated that represents the transition block. In approach III, the transition blocks are included directly.

The *table-controlled access* implements the access procedure by means of a table (matrix) that contains the addresses of the transition blocks. The input parameters are, for instance, states and incoming events. As in Estelle the selection is usually based on three clauses (*from*, *when*, *provided*) a three-dimensional matrix is needed. Such a matrix gets soon inefficient due to the large number of values in the *provided*-clauses to be tested. For that reason, a combination of the programmed and the table-controlled access is usually preferred. The programmed selection is used for the *provided*-clauses, whereas for the other clauses the table-controlled method is applied.

A comparison of the two access methods is given in Figure 3. It is based on the implementation of the experimental protocol [Held 94]. The implementation uses the simplified asynchronous interface (see Section 4.3) and the activation algorithm III. The *provided*-clause is not taken into account as both implementations use the programmed access for it. Figure 3 indicates the absolute response time per PDU related to the average number of decisions needed to access to the program code. It shows that from 4 decisions upwards the table-controlled method is superior to the programmed one because the number of decisions has no influence on the response time. In real protocols more than 4 decisions are usually needed.

Fig 3: Comparison of the programmed and the table-controlled access

4.3 Interfaces

The process model chosen for the implementation determines the type of the interface. In the server model, asynchronous interfaces are used as a rule for the realization of the interlayer communication. They are implemented by paired queues. This approach is used in many Estelle implementations [NBS 87], [Sije 90], [Vuon 88] whereby Estelle does not distinguish layer and module interfaces. We have implemented two types of the asynchronous interface in the EECT. The first type implements a queue for each interaction point (*individuel queue*) or for a group of interactions points (*common queue*). The other type uses only one central queue for all interactions. The goal of the second approach (simplified interface) has been to decrease the overhead of the queue management.

The comparison of the two realizations for the implementation of the LLC-3 protocol (see Fig. 4) points out that the simplified interface increases the absolute response time per PDU by about $100\text{ }\mu\text{s}$, i.e. 8,3 percent. The share of the interface routines on the total response time decreases by about 2.

Fig. 4: Comparison of the two interface realizations

4.4 Implementation environment

The implementation environment constitutes the virtual machine on which the protocol is implemented. It comprises those functions needed for the realization of the protocol but which cannot be directly derived from the logical model. The implementation environment connects the derived protocol software with the operating system, with the adjacent layers, in part with the hardware (layer 1 and 2), and with the network management. Typical functions of the implementation environment are the scheduler, routines for the realization of the module interfaces, routines for the management of the module instances, time control, and other auxiliary routines. For computer-aided implementations, a separate implementation environment is usually provided.

The implementation environment is often called the hand-coded part of a computer-aided implementation. The relation between the hand-coded and the automatically generated part of the implementation is one of the criteria for the efficiency of this approach [Cimi 88], [Vuon 88]. This is correct insofar as the functions of the implementation environment have to be manually provided once. When the environment is implemented and tested it constitutes a reliable component of the implementation that can be reused in other implementations. Thus, the automation degree of the approach is relatively increased [Koen 90]. (The discussion about the relation, however, has to take the concrete protocol into account as the portion of the implementation environment remains unchanged whereas the portion of the generated code depends on the size of the specified protocol.)

The share of the hand-coded part on the runtime of the protocol is, however, much larger than in the static relation. That's why the implementation of the hand-coded functions requires much care as well. Figure 5 shows this share for an implementation of the LLC-3 protocol. For this implementation, we used the simplified asynchronous interface, the selection approach III and the programmed access as discussed in the previous sections.

The measurement gives evidence that the runtime share of the hand-coded part is higher than its portion on the code. It is almost 50 per cent in this experiment. Besides, this is an optimal relation because we have chosen the most effective scheduler realization. For the scheduler approach I and II, the relation would be worse (see Figure 2). The measurement shows further that among the hand-coded routine the share of the scheduler is much higher than the others although the interface routines are called most.

Fig. 5: Share of the hand-coded and the automatically generated part on the runtime of a LLC-3 implementation including the frequency of the routine calls

5. A variable implementation model

The discussion in the previous section has shown that design decisions for the implementation model can have a great impact on the efficiency of the final implementation. As these decisions are made during the design phase of the tool they cannot take into account the implementation contexts of their future applications. This limits the efficiency of the final implementation and forces revisions to the generated code including the implementation environment. This again restricts the automation degree and the application of the tool.

To enable the implementation tool to adjust to the concrete implementation context, they should support a variable implementation model. Figure 6 shows the variable implementation model we have implemented for the EECT. It provides options for the realization of the certain components of the model. These alternatives are marked by the finger symbol. By combining these alternatives the implementor can adjust his implementation model to the concrete context. The implementation environment is also variable. Depending on the implementation model selected its size varies between 10 and 20 KByte. The environment is available for UNIX.

For the configuration of the implementation model, a program library has to be composed that is included into the EECT. Currently, we are implementing a separate user interface.

6. Conclusions

In this paper, we have discussed some approaches to increase the efficiency of computer-aided implementation techniques. We have pointed out that the implementation model supported by implementation tools has a crucial effect on the efficiency of the generated code and the auto

Fig. 6: Variable implementation model of the EECT

mation degree. Existing transformers for FDTs use in principle a fixed implementation model. Future tools should be capable to adjust to the context of the implementation in the best possible way. In addition, further work is needed to improve the technology by using new principles such as the mapping on the activity-thread process model, the integration of several layers, and the use of parallel implementation structures. Moreover, the requirements of FDTs and related specification styles to support efficient implementations should be investigated.

References

- [Agga 85] Aggarwal, S; Kurshan, R.P.: Automated Implementation from Formal Specification. In Yemini, Y. et al. (eds.): PSTV IV, North Holland, 1985, pp. 127-136.
- [Chan 93] Chanson, S.T. et al.: On tools supporting the use of formal description techniques in protocol development. *Computer Networks and ISDN Systems* 25(1993), pp. 723-739.
- [Cimi 88] Ciminera, L.; Demartini, C.: A Practical Study of Semi-automatic OSI Protocol Implementation. *Software Practice and Experience*, 18(1988)3, pp. 255-277.
- [FoHe 92] Foedisch, R.; Held, T.; Koenig, H.: A Protocol Development Environment Based on Estelle. *Proc. INDC'92*, Helsinki, North Holland, 121-141.
- [Held 94] Held, Th.: Practice of the Computer-aided Protocol Implementation. Ph.D. Thesis. TU Magdeburg, Fak. Informatik, 1994 (in German).
- [ISO 88] ISO: Logical Link Control Type-3 Operation; Acknowledged Connectionless Service. ISO/IEC JTC 1/SC6, 1988.
- [ISO 89] ISO IS 9074: Estelle - A Formal Description Technique Based on an Extended State Model, 1989.
- [Koen 90] Koenig, H.: Experience in Computer-Aided Protocol Implementation. In Csaba, L. et al. (eds.): *Computer Networking*, North Holland, 1990, pp. 385-393.
- [Nagl 90] Nagl, M.: *Softwaretechnik: Methodisches Programmieren im Großen*. Springer-Verlag, Heidelberg, 1990.
- [NBS 87] U.S. Department of Commerce National Bureau of Standards. *Internals Guide for the NBS Prototype Compiler for Estelle*. Final Report, Report No: ICS/SNA - 87/4, 1987.
- [Prof 91] Turbo Profiler - Benutzerhandbuch. Borland International, INC. München, 1991.
- [Sije 90] Sijelmassi, R.; Strausser, B.: NIST Integrated Tool Set for Estelle. In *Proc. FORTE'90*, Madrid, 1990.
- [Svob 89] Svobodova, L.: Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications*, 7(1989)7, pp. 1115-1130.
- [Vuon 88] Vuong, S.T. et al.: An Estelle-C Compiler for Automatic Protocol Implementation. In Sabnani, K. (ed.): *PSTV VIII*, North Holland, 1988, pp. 387-398.
- [Wats 87] Watson, R.W.; Mamrak, S. A.: Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices. *ACM Trans. on Computer Systems*, 5(1987)2, pp. 97-120.