# Protocol Implementation with Estelle — from Prototypes to Efficient Implementations[1]

J. Thees
University of Kaiserslautern
Germany
thees@informatik.uni-kl.de

## Abstract

*Automatic generation of implementations directly from formal specifications is an important factor for the attractiveness of formal methods in practical software development. Important requirements from a practical point of view are correctness, efficiency and simple integration into pre-existing environments.*

*In this paper we present the eXperimental Estelle Compiler (XEC). Together with a set of performance analysis tools and benchmark specifications it builds a platform for implementation and optimization experiments with the FDT Estelle. Furthermore, we discuss different optimization approaches and extensions that we are currently implementing based on XEC. This paper describes ongoing work.*

## Keywords

*FDTs, Estelle, automatic code generation, efficiency, optimization, Estelle extensions*

## 1 Introduction

A key factor for the attractiveness of formal methods in practical software development is the generation of correct and efficient implementations directly from formal specifications, since it makes it possible to save (or at least to simplify) the costly and error prone step of manual coding. But automatically generated implementations suffer from specific problems that hamper a wide spreading in practical software development. Some of these problems arise from the limited expressiveness and high abstraction level of FDTs, while others are caused by the limitations of automatic code generators in the creation of optimized implementations and their integration into preexisting environments.

Several approaches for an efficient (semi-) automatic implementation of formally described protocols have been proposed in the past years, but only few have been implemented and evaluated quantitatively under realistic conditions. One cause for this deficit may be the absence of a freely available experimentation platform that supports implementation and optimization experiments at a high abstraction level.

1. This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant Go 503/4-1.

In this paper we present such a platform designed for experiments with implementation and optimization methods for the FDT Estelle [DeBu89, ISO97]. Our goal is the practical evaluation of implementation and optimization methods on three different levels:

- the code generation,
- specification style, and
- the FDT itself.

This paper describes ongoing work.

## 2 The eXperimental Estelle Compiler (XEC)

To build a platform for the *testing, performance-evaluation, and optimization of implementation methods* for Estelle, we have developed the *eXperimental **E**stelle **C**ompiler* (*XEC*, [ThGo98a]). XEC is a fully operational and well tested Estelle compiler. It is called “experimental” because it is intended as a platform for experiments with implementation and optimization methods.

The implementation generator builds upon the Estelle compiler front-end PET [SiSt93] and translates Estelle[2] specifications into object-oriented C++ code. The strictly object oriented modeling of a specification’s structure permits a very flexible execution model, supporting extensive *static and dynamic optimizations*. Most of the dynamic optimizations can be implemented inside the object oriented run-time library for the generated implementations. This reduces the complexity of the code generator itself, and therefore also the risk of errors due to complex compiler optimizations.

Despite of their high abstraction level and flexibility, the automatically generated implementations are enabled to be efficient, as a result of performance oriented coding of the run time library and generated code, and especially by the selective use of inline code.

As an example for the high abstraction level of the generated code, Figure 1 sketches the representation of the specification structure as a system of *nested C++ classes* (i.e. as classes defined as members of other classes). This technique does not only preserve the specification structure, but it also makes it possible to project the naming scheme of Estelle into C++ in a natural and unique way, because it defines a hierarchical name space in which every (named) definition has a globally unique *qualified name* (e.g. `SPEC_S::BODY_A::BODY_B`). Obviously, there is no need to create artificial extensions for identifiers (e.g. consecutive numbers like in code generated by EC or DINGO) to make them unique[3].

```
specification S;
    body A
        body B
            end;
        end; {body A}
    body C
        end;
    end. {specification S}
```

XEC →

```
class SPEC_S {
    class BODY_A {
        class BODY_B {
        };
    }; /* class BODY_A */
    class BODY_C {
    };
}; /* class BODY_S */
```

**Figure 1. Principle of XEC’s Representation of Estelle Specification Structure**

---

2. XEC also supports some Estelle extensions, e.g. the independent translation of open systems and importing environments described with the Estelle extension “Open Estelle” [ThGo98b].
3. The *classification prefixes* created by XEC (e.g. `SPEC_`, `BODY_`, `TYPE_`, `VAR_`, `CONST_`, `TRANS_`) are mainly used to increase the readability of the generated C++ code and could be omitted.

Another example for the high abstraction level and the support of dynamic optimizations is the representation of *Estelle transitions as C++ classes*. During the instantiation of a module, instances of these transition classes are created[4] and collected in a special list inside the module instance. In order to model the concrete transitions, the generated transition classes simply overload virtual functions of their base class for every transition clause stated in the specification text.[5] This leads to a 1:1 representation of the underlying specification (see Figure 2). This representation of transitions as concrete instances of a polymorphic base class builds the basis for many sophisticated dynamic optimizations: each transition can keep its own statistics to support *dynamically selected optimization heuristics*, temporary *inactive* (i.e. non-selectable) *transitions* can be removed dynamically from the transition lists (until certain enabling events occur), *varying selection mechanisms* can be used to adopt to the current local execution situation, and so on.

```
trans
    from s1
    provided n <= max
    name test:
        begin
        n := n+1;
        end;
```

XEC →

```
class TRANS_test : SimpleTrans {
        public:
            bool checkFrom(int nState)  {return nState == STATE_s1;}
            bool checkProvided()   {return VAR_n <= CONST_max;}
            void exec() {VAR_n = VAR_n + 1;}
}; /* class TRANS_test */
```

**Figure 2. Principle of XEC's Modeling of Estelle Transitions**

## 3 Performance Valuation of Protocol Implementations

An important aspect for the quantitative valuation of implementation and optimization experiments is the support of detailed and application related performance measurements. For this purpose we developed two benchmarks, a *synthetic benchmark* for detailed experiments with special implementation aspects and an *application benchmark* for practical overall performance measurements [Dah96].

Our application benchmark is based on a relatively large Estelle specification[6] (about 7900 lines) of the eXpress transport protocol (XTP 4.0, [XTP95]). This specification was not developed or optimized with regard to performance aspects and therefore represents a typical high-level protocol specification. It covers some very interesting implementation aspects like *dynamic reconfiguration*, deeply structured module hierarchies, complex protocol mechanisms, and the transmission and buffering of *variable sized data packets*. Furthermore, the use of this application benchmark allowed us a direct and realistic comparison [ThGo97a] of different automatically generated implementations (generated with PET-DINGO [SiSt93], EC [RiCl89] and XEC [ThGo98a]) and a hand coded implementation (SandiaXTP [SNL95]).

Besides these benchmark tests, which already deliver overall performance data for typical application situations, we have developed the tools EGD and PATO for detailed performance evaluations. PATO (Performance Analysis Tool [Pen96]) can gather timed execution traces of Estelle implementations and interactively evaluate these data with a graphical front-end. It is possible to monitor global activities like the processing of data packets on their way from end to end, as well as the time consumption of local events (down to the level of single statements

4. ANY-clauses of transitions are expanded by multiple instantiation of the respective transition-classes.
5. Transition clauses omitted in the specification text (i.e. implicit default clauses) are also omitted in the generated C++-code, since their default semantics is represented by the transition base class implicitly.
6. The original Estelle specification of XTP 4.0 that was kindly made available by Stanislaw Budkowski (INT, France) and Octavian Catrina (PU Bucharest, Romania).

or transition clause evaluations). Despite this subtle tracing resolution, the overhead of the monitoring can be kept under control by the possibility to determine the generation of traces precisely.

Another Estelle related tool is EGD [Wen98], a JAVA based graphical debugger front-end designed for the visualization and control of Estelle implementations generated by XEC. It is possible to step through the execution of a specification, visualize local state components of modules, set breakpoints to the selection of transitions, and engage in the transition selection process (similar to the basic functions of EDB [RiCl89]). The main tasks of EGD concerning performance evaluations is the user-controlled reaching of states of execution and the interactive selection and restriction of execution traces for PATO.

# 4 Optimization Approaches for Estelle Implementations

Based on our implementation platform introduced in the previous sections, we are currently evaluating different optimization approaches. These cover both, already proposed (e.g. [Go+96], [Fis96], [Bre97], [HMK97]) and several new techniques. These optimization experiments will cover the code generation itself, as well as the specification-style and even the expressiveness of Estelle itself. In this chapter we will discuss some of the optimization approaches we are currently evaluating.

From our experiments we learned two decisive factors that decrease the performance of automatically generated Estelle implementations in direct comparison to manual coded implementations: *transition selection* and *data transport*.

## 4.1 Transition Selection Process

Because of the Estelle-specific synchronization and selection rules a large amount of time during the execution of Estelle-conforming implementations is spent in the transition selection process. E.g. during the transmission of small data packets with a non-optimized Estelle-implementation of XTP (generated by XEC, straight selection process), about 87% of the total execution time is spent inside transition selection phases, because on the average about 161 transitions have to be tested to find one fireable. The introduction of a first simple (global) optimization of the selection method (“inactive module tree”, see below) nearly doubles the performance of the implementation (selection time 74%, test-to-fire ratio 61:1) and allows XEC to catch up with implementations generated by EC [RiCl89]. Nevertheless both implementations are about a factor 2.3 slower than the manual implementation SandiaXTP [ThGo97a].

Obviously there is still considerable room for optimizations concerning the selection process. One aspect are table oriented selection methods for *module-local transition selections*, like the ones already implemented in EC. Our experiments show that table oriented control state and input evaluation can speed up the local transition selection by factor 2 and more (especially with common queue buffers). Another class of optimizations for local selections we are currently going to implement are based on the object representation of transitions in XEC: these transition objects can be removed from the list of transitions to be evaluated, as long as one of their necessary enabling conditions is not fulfilled, i.e. the transition is *inactive*. This technique is only suitable to conditions that are enabled and disabled rarely (e.g. occurrence of disconnect messages or error flags) and whose enabling can be perceived effectively (e.g. messages or simple variables).

A class of optimization heuristics that can improve the *global transition selection* (i.e. the selection of modules containing fireable transitions) is the effective guessing of modules that *may* (or *cannot)* contain fireable transitions. One of the simplest of these methods is the "*inactive module tree*" optimization already noted above. It tries to detect maximum sub-trees of the module instance tree, which did not contain fireable transitions in the last selection phase. The modules in such a module-subtree can be excluded from further selection steps, until another object interacts with it, or a delay-timer inside the module-subtree expires.

Another promising optimization is based on the observation that a module that receives a message into a previously empty FIFO-buffer nearly always contains a fireable transition to consume this message. Consequently the selection (and thus the execution-process) just has to *follow the data- (i.e. message-) flow* in order to find the next module with fireable transitions. Together with an effective local transition selection this server-controlled selection-heuristic can reach the same efficiency as dedicated *activity thread* implementations: after a transition has sent a message, the "successor transition" (which will consume the message) can be selected with a minimum number of transition tests — in most cases only one transition test will do. Different from the incisive stylistic restrictions necessary for an activity thread implementation [HMK97], our optimization method is also applicable for specifications that can't avoid father-son-priority (e.g. the XTP specification with its dynamic reconfiguration), even though such constellations cause a slight increase of the selection overhead. The application of the "*asynchronous process*" extension [Go+96] can eliminate this problem in most cases.

There are several promising optimization techniques, but most of them are only suitable for specific execution conditions. If these conditions are not fulfilled, most optimization methods decrease the performance of the implementation significantly. Generally it is very difficult to determine statically, which optimizations are most suitable for a given specification, especially if the behavior of a system changes during its execution (e.g. changing packet-size, error-rate, delay-time, connection-mode, and so on). The flexible structure of the implementations generated by XEC and the dynamic selection between different selection mechanisms makes it possible to switch between different local and global optimization heuristics and methods, in order to adopt to current execution situations. Together with user-defined *hints* and saved results from *profiling traces*, these dynamic adaptive optimizations can improve the selection process considerably, as long as their run-time overhead can be kept under control.

### 4.2 Data Transport

A comparison between implementations generated from the Estelle specification of XTP and the manual implementation SandiaXTP shows two other conceptual problems of Estelle concerning an efficient automatic implementation: the specification of *variable sized data objects* and the *copy semantics* of data transmission between module instances.

In SandiaXTP [SNL95] variable sized data packets are placed in a shared memory buffer and they are referenced by scatter gather vectors. This technique makes it possible to transmit and re-frame packets through the whole XTP-PM without any copy operations.

The Estelle specification of XTP represents the packet payload as a fixed sized array of bytes (0..255), which is resent at least 5 times inside the XTP-PM. The size of this array limits the maximum packet size that can be sent with one transmission. But Estelle has no means to describe dynamically sized structures or the effective (i.e. used) occupation of a partially occupied array, so the complete array has to be transmitted. Furthermore current Estelle compilers implement data transport by at least one copy operation. So the complete array has to be copied 5 times. This has considerable impact on the performance of the generated implementation, if practical relevant packet limits (has to be fixed at compile time) are used: the total transmission of a 10 byte packet is about 3 times slower (EC and XEC), if the packet limit is raised from 10 bytes to 10,000 bytes.

One possible solution for this problem is the definition of the binary packet payload as incompletely defined data type with "primitive", implementation related access functions[7]. These manually coded functions can store the data in a shared memory buffer similar to SandiaXTP. This method, together with a strictly implementation oriented specification style and an optimizing code generator (EC), enabled in [CaNo98] a generated implementation of XTP[8] to catch up even with the performance of the manual implementation SandiaXTP. The main disadvantage of this solution is the need for manual modifications or extensions of the generated code, which require detailed knowledge of both, the operations of the compiler and of the Estelle specifications, in order to conserve the originally intended semantics, especially to prevent the risk of a violation of the copy semantics of Estelle's message transmission.

Alternative solutions we are also investigating are Estelle-extensions in order to support shared data between tightly coupled modules (e.g. modules inside subsystems), explicit transmission of data by-reference, and data objects of variable size ("binary large objects"). These extensions allow a specifier to describe the data transport tasks more detailed and problem-related and additionally enable automatic code generators to create efficient code without manual intervention.

# 5 Outlook

The future experiments will have to show, what amount of performance gain is possible with improved automatic code generators, implementation oriented specification styles, manual modifications of generated code, or even extensions of the expressiveness of current FDTs. Perhaps one day the automatic code generation from formal specification will reach a level that makes manual implementations as unusual as assembler coding in today's software development. But code generation could become at least a considerable alternative to manual coding in practical software development.

# 6 References


[Bre97] Bredereke, J.: *Communication Systems Design with Estelle — on Style, Efficiency, and Analysis*, PhD thesis, Univ. of Kaiserslautern, Shaker Verlag, Aachen, Germany, 1997

[CaNo98] Catrina, O., Nogai A., *On the Improvement of Estelle Based Automatic Implementations*, accepted for FORTE/PSTV'98, Paris, France, 3-6 November 1998

[Dah96] Dahl, S.: *Performance Comparison of Protocol Implementations* (in german), Master thesis, Univ. of Kaiserslautern, Dept. of Comp. Sci., 1996

[DeBu89] Dembinski, P., Budkowski, S.: *Specification Language Estelle*, in: M. Diaz et al. (eds.), *The Formal Description Technique, Estelle*, North-Holland, 1989, pp. 35-75

[Fis96] Fischer, S.: *Formale Spezifikation und Implementierung von Hochleistungsprotokollen*, PhD thesis, Univ. of Mannheim, Shaker Verlag, Aachen, Germany, 1996

[Go+96] Gotzhein, R., Bredereke, J., Effelsberg, W., Fischer, S., Held, T., König, H.: *Improving the Efficiency of Automated Protocol Implementation Using Estelle*, Computer Communications, 19 (1996), pp. 1226-1235

[HMK97] R. Henke, A. Mitschele-Thiel, H. König. *On the Influence of Semantic Constraints on the Code Generation from Estelle Specifications*, Joint Int. Conf. FORTE/PSTV'97, Osaka, Japan, Chapman-Hall, Nov. 1997

[ISO97] ISO/TC97/SC21: *Estelle - A Formal Description Technique Based on an Extended State Transition Model*, ISO/TC97/SC21, IS 9074, 1997


7. With "functions" we also refer to "procedures" in this context.
8. The XTP specification used in [CaNo98] is an extensively optimized (i.e. substantially modified) version of the XTP specification used in our experiments noted in Sections 3 and 4, which was not specified under performance aspects. Consequently the performance data are currently not directly comparable.

[LFV95] Lallet, E., Fischer S., Verdier, J.-F.: A New Approach for Distributing Estelle Specifications, Formal Description Techniques VIII, Chapman & Hall, London, 1995

[Pen96] Penner, H.: *Implementation of a Software Monitor for the Analysis of Automatically Generated Protocol Implementations* (in german), Master thesis, Univ. of Kaiserslautern, Dept. of Comp. Sci., 1996

[RiCl89] Richard, J. L., Claes, T.: *A Generator for C-Code for Estelle*, in: M. Diaz et al (eds.), The Formal Description Technique Estelle, North-Holland, 1989, pp. 397-420

[SiSt93] Sijelmassi, R., Strausser, B.: *The PET and DINGO tools for deriving distributed implementations from Estelle*, Computer Networks and ISDN Systems 25, 1993, pp. 841-851

[SNL95] Sandia National Laboratories: SandiaXTP — An Object-Oriented Implementation of XTP 4.0 Derived from the Meta-Transport Library: User Manual, Release 1.4, Livermore, California, USA, October 1995

[ThGo97a] Thees, J., Gotzhein, R.: *Performance Evaluation of Automatically Generated Protocol Implementations* (in german), 9. ITG/GI-Workshop “Messen, Modellieren und Bewerten von Rechen- und Kommunikationssystemen”, Freiberg, Germany, 1997

[ThGo97b] Thees, J., Gotzhein, R.: *Generation of Efficient Protocol Implementations - an Experimental Code Generator for Estelle and its Application to XTP*, 6th Open Workshop on High Speed Networks, Stuttgart, October 8-9, 1997

[ThGo98a] Thees, J., Gotzhein, R.: *The eXperimental Estelle Compiler - Automatic Generation of Implementations from Formal Specifications*, in: M. Ardis (Edt.), *Proceedings of The 2nd Workshop on Formal Methods in Software Practice (FMSP’98)*, Clearwater Beach, Florida, USA, March 1998

[ThGo98b] Thees, J., Gotzhein, R.: *Open Estelle - An FDT for Open Distributed Systems*, accepted for FORTE/PSTV’98, Paris, France, 3-6 November 1998

[Wen98] Wenz, M.: *Design and Implementation of a Portable Graphical User Interface for the Visualization and Control of the Execution of Estelle Implementations*(in german), Master thesis, Univ. of Kaiserslautern, Dept. of Comp. Sci., 1998

[XTP95] XTP Forum, *Xpress Transport Protocol Specification*, XTP Rev. 4.0, XTP Forum, Santa Barbara, USA, 1995