16
Application Level Framing and Automated Implementation

Christophe Diot[1], *Isabelle Chrisment*[1], *Antony Richards*[2]

[1]*INRIA Sophia Antipolis, FRANCE*
christophe.diot@sophia.inria.fr; isabelle.chrisment@sophia.inria.fr

[2]*UTS Sydney, AUSTRALIA*
antony@ee.uts.edu.a

Abstract

New concepts such as the Application Level Framing (ALF) have been proposed to make network protocol implementations more efficient and to give the application programmer greater control over the data transmission. This paper describes early experiments with automated design and implementation of application-specific communication protocols based on the formal specification of the application using ESTEREL. A comparison is made between a hand coded JPEG player and its automated equivalent. The results show that the automated approach creates a better integrated implementation with the same level of performance.

Keywords

Efficient Communication Architectures, ALF, Integrated Implementation, Formal Specifications, User Level Protocols.

1.0 INTRODUCTION

The rapid evolution of networking and the multiplication of new applications re-emphasizes the importance of the efficient (i.e. flexible and performant) communication supports. Implementations must be able to take maximal advantage of details of application-specific semantics and of specific networking environments to be performant. In other words, the application needs to have more control over data transmission, as proposed in the Application Level Framing concept. ALF [Clark90] is a design principle in which the communication subsystem is able to process data in chunks of application-specific size. Such control can be obtained only by tailoring the communication facilities (or protocols) to the application characteristics.

R. Puigjaner (ed.), *High Performance Networking*

Efficiency of the implementation can also be discussed independently of the protocol used. It is well accepted today that, if a layered architecture is a good abstract frame for specification, it is a penalty with respect to implementation performance [Crowcroft92]. A layered architecture introduces undue redundancies in interfaces and data transfers. An efficient implementation would "integrate" communication facilities in a more horizontal scheme. The realization of ALF in such a non-layered architecture will produce an implementation with a high level of integration, which will be difficult to implement manually. The traditional empirical methodology, which is mostly "intuition based on experience", will not be able to scale up effectively and to produce highly integrated implementations for the new generation of applications.

To be able to design efficient implementations of communication support tailored to application characteristics, it is claimed that the development process of communication support must largely be automated (in a formal framework). A formal automated design process also allows the correctness of the communication to be checked both in terms of reliability and security, and even the efficiency can be determined given set of constraints (like QoS requirements).

The design of an integrated environment for automatic implementation of communication supports is necessarily based on a formalism to be introduced as early as the application definition step. As a result it should yield an automatic (or mostly automatic) generation of the end-to-end transmission control mechanisms integrated into the application specification before implementation. The performance speed-up is expected from more integration (minimizing interfaces), from the application of ALF, and also from the various optimizations made possible by the automated approach.

This paper describes early experiments with the automated design and implementation of application-specific communication protocols based on the formal specification of the application. A JPEG player [Wallace90] is used as example. Starting from its formal description, a "Protocol Compiler" automatically integrates communication facilities to the application following the ALF concept. The second stage of the Protocol Compiler (which is today partially automated) produces a high performance implementation of the application with its communication facilities.

The rest of the paper is organized as follows. Section 2 presents the benefits of the formal approach and introduces the ESTEREL language which is the formalism retained for the realization of the Protocol Compiler. In section 3, the Protocol Compiler is reviewed. Its principle is described as well as the way communication facilities are integrated into the application formal specification. Section 4 describes the early experiments with the prototype Protocol Compiler. A JPEG player was specified in ESTEREL. On one side, an hand-coded ALF implementation was written in C; on the other, the ESTEREL specification was processed through the Protocol Compiler to produce automatically another implementation. Performance evaluation, and comparison is provided. The paper concludes on the analysis of this approach, and on the possible extensions of the Protocol Compiler.

Note that this paper does not address Quality of Service problems. Tailoring is only done considering the nature of the services required by the application. Quality of transmission will be studied later, through the concept of Network Conscious Applications [Diot95].

2.0 FORMAL APPROACH

2.1 Problems solved

The current approach to distributed applications is the Remote Procedure Call (RPC) [Nelson81] concept. RPC is a very simple client/server model that proposes the same principles as local procedure call, but using a communication support. This communication support, which is based on a synchronous request/response protocol is not efficient enough for multimedia and time constrained applications. The Protocol Compiler is a preliminary step in the design of a new generation RPC model, where a distributed application can specify its own communication requirements to be associated to a dedicated transmission control protocol. The control and synchronization aspects of an application are formally specified using ESTEREL. This specification is then used by the protocol compiler to integrate transmission control facilities to the application, and to generate the client and server stubs. The advantages of this approach are the following:

- Like in the classical RPC, the transmission facilities remain transparent to the application designer.
- The control of the transmission is given to the application. The communication facilities are integrated to the application from which they have been tailored.
- The implementation is designed automatically, starting from the application formal specification. It guarantees the implementation matches exactly with the original application specification. It is also easier to prove that the communication system provides the application minimal end-to-end transmission control facilities.
- The formal approach allows the systematic use of optimization techniques such as optimizing the protocol control automaton, discovering the most frequently used path, inlining code, etc. These optimizations might lead to better performance than with hand-coded implementations.
- There is no interface between the application and the transmission control mechanisms, which are implemented as part of the application. Consequences are increased performance and improved flexibility.

The approach outlined in this paper is original as it starts from the application specification, and as the end-to-end transmission control facilities (also called communication subsystem in this paper) are integrated to the application specification before the automated implementation. Flexibility is allowed by the integration of the transmission control mechanism within the application,and by the implementation in the user space of the host computer.

2.2 ESTEREL and its development environment

ESTEREL is an imperative language belonging to the family of the ***Synchronous Reactive Formalisms*** [Berry92]. ESTEREL is *not the only* candidate language for protocol engineering; but it is a good compromise considering previous experiments we had with it in protocol design [Castel94][Chrisment94a]. It contains particular features we are looking for, such as reactivity or broadcast. A complete analysis of ESTEREL for communication protocol development is given in [Diot94]. Some of the ESTEREL features discussed in [Diot94] are:

- Modular parallelism. A given protocol can be obtained as a collection of smaller modules being selected (or not) and properly combined. The sequential implementation produced by the ESTEREL compiler is determinist.
- Expression of Finite State Machines. ESTEREL is a control description language; that means only the control automaton of the application and its communication support can be described in ESTEREL. A data description language like XDR [Corbin91], ASN1 [ISO87b], or C (as used in this paper) has to be associated with ESTEREL.
- Readability. The JPEG specification written in ESTEREL is only one page of code for the client side, and another page for the server side. ESTEREL syntax, which is close from Pascal and CSP [Hoare79] makes ESTEREL descriptions very easy to read.
- Time handling. There is no special type or signal to describe time in ESTEREL. Time is considered as any other external event, and must be described by an external signal. Time in ESTEREL is multiform: any signal may be processed as an independent "time unit", so that the time manipulation primitives can be used uniformly for all signals.

In addition, the ESTEREL environment contains various tools (compiler, parallel debugger, graphical simulator, proof systems, optimizers) [Berry92][Meije94]. All these tools concur to provide a very powerful environment for the development of complex reactive systems like communication protocols.

The Protocol Compiler is fully compatible with the ESTEREL compiler; which means the information that has to be added to the ESTEREL specification for parsing purpose has an ESTEREL syntax. The major advantage is the application specification can be compiled, verified, and analyzed by using the ESTEREL environment tools. Further, after protocol integration, the ESTEREL syntax is still respected. The application parser consequently processes an ESTEREL specification of a distributed application and produces an other ESTEREL description where the communications facilities have been integrated to the original specification.

3.0 THE PROTOCOL COMPILER

The prototype of the Protocol Compiler is made of two distinct parts, i.e a parser and an implementation generator (Figure 1).

The application designer creates the application software and data structures using C, XDR or ASN.1. The application specification is written in ESTEREL. This specification is then parsed to produce the *integrated specification.* The *parser i*ntroduces protocol functions using templates that perform the protocol mechanisms. The protocol functions exist within an ESTEREL library. The integrated specification is then compiled using the *implementation generator.* The resulting C code is itself compiled and linked with the protocol function library and with the application software in order to be integrated in its execution environment.

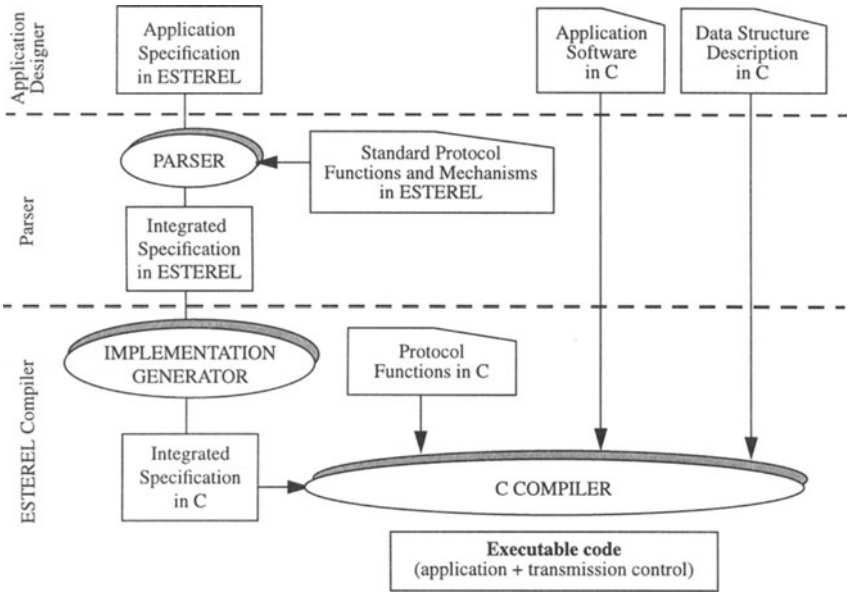

Figure 1: the architecture of the Protocol Compiler

3.1 The parser

The application parser is the first stage of the protocol compiler. It integrates dedicated communication facilities to the ESTEREL specification of a distributed application (Figure 2). The module resulting from application parsing is also an ESTEREL module.

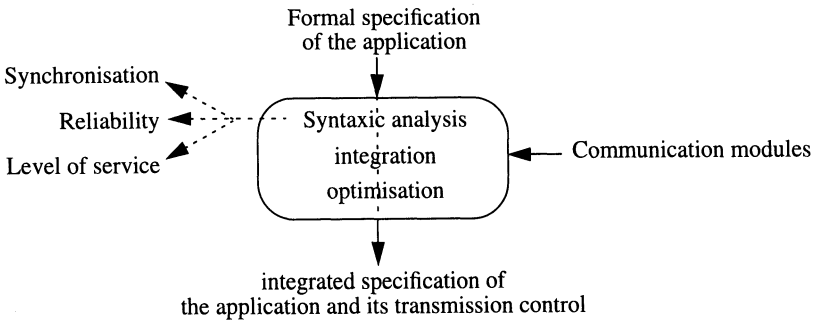

Figure 2: structure of the parser

The formal application specification is made of two types of information:

- The ESTEREL description of the application behavior. The parser extracts from this description information concerning the structure of the application, the possible synchronization points and parallelism between the different modules that compose the description. These synchronization points will be used at the implementation level, to design the most efficient implementation.
- Additional information is introduced in the application to describe application characteristiques that cannot be deduced from the formal specification, i.e. the level of reliability required for the transmission of Application Data Units (or ADUs). An ADU is defined as the smallest unit of data that the application can handle out of order. This communication-related information is added in a syntax that does not modify the behavior of the original application description, and that is easy to understand by any reader. As previously stated, the additional information has an ESTEREL syntax. The type of service required is described by a set of keywords expressed in ESTEREL by a list of local signals. Keywords can be "selective_retransmission", "flow_control", "checksum", "encryption", etc. This list of local signals encapsulates the modules to which it refers within a *present* instruction.

Figure 3 illustrates an example of an application specification where retransmission is done on timeout and where selective acknowledgment is used for error control. The left code gives the specification that should be written by the application designer. *Reliable* and *select_retrans* are two keywords. The encapsulation within the *present* instruction has no effect on the ESTEREL compiler. The parser recognizes input signals that have to be processed through a transmission control protocol by the *Remote_* prefix added by the designer to concerned signals. The ESTEREL code on the right side of the figure shows how the original specification has been processed by the application parser to integrate the corresponding transmission control facilities to the application specification. The *present* instruction has disappeared; the module now waits simultaneously on two input events:

- Remote_ADU which now will be processed in parallel to the evaluation of the acknowledgment to be sent.
- A timer on which timeout selective retransmission has to be done.

```
present [reliable and select_retrans] do
    loop
        await Remote_ADU;
        call Process_ADU ()()
    end loop
end present
```

application specification

```
loop
    await
        case Remote_ADU do
            call Process_ADU ()()
            ||
            call Process_ack ()()
        case Timeout do
            call Process_selective_retrans ()()
    end await
end loop
```

description after protocol integration

Figure 3: additional information syntax

The parser is able to determine what type of protocol is required and where this protocol has to be integrated into the application specification. The communication functions and mechanisms to be integrated are found in a library and are written too in ESTEREL.

The application parser also defines the structure and contents of the protocol headers used (based on the communication mechanisms added to the application specification).

The ESTEREL specification of the application is required (instead of a simpler formalism) to define the transmission control functions and to integrate the protocol because the Finite State Machine (FSM) of the application can be used for the control of the transmission on the network. The use of a simpler formalism (for example a list of application requirement parameters) would limit the efficiency of the application parser because:

- the transmission control automaton can be directly mapped into the application control automaton for maximum integration (integration is not possible in case of a list of parameters).
- Information on possible parallelism and module synchronization is revealed from the application specification architecture.

3.2 The implementation generator

The second stage of the Protocol Compiler consists of the transformation of the integrated application (which is an ESTEREL description) into an executable module (written in C code). This automated implementation mostly relies on the ESTEREL compiler, which provides all the facilities for such a transformation (Figure 4). The fact that a dedicated implementation generator is not used only limits possible optimizations and code inlining..

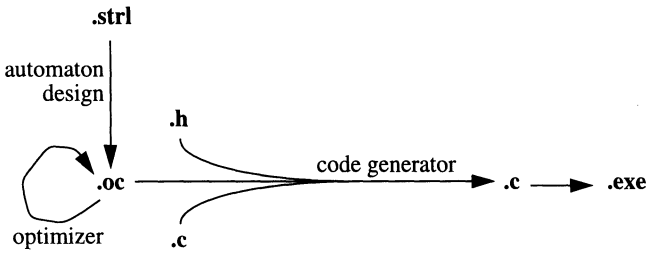

Figure 4: modular architecture of the ESTEREL compiler

The ESTEREL compiler processes the integrated application description in two steps:

- The integrated application FSM is first built, and then optimized. During this first step, the original ESTEREL language is transformed in an intermediate format named oc.
- Oc code is then compiled, and linked with C libraries that contain the definition of the data types and the data manipulation functions that were used in the ESTEREL description. Note that various C modules can be used to optimize the implementation performance in

different host environments. These C libraries also contain modules that describe the interface between the ESTEREL specification and the execution environment.

The code generator produces a sequential implementation of the ESTEREL module, even if parallelism is used in the application description. The code generator is designed to optimize the code produced, and to resolves all the problems that could occur because of shared resources.

4.0 IMPLEMENTATION OF A JPEG PLAYER

To illustrate the concepts presented in this paper, a client-server based JPEG [Wallace90] image player have been implemented. The client is a menu based program that allows interaction with a user. The user can request an image to be displayed, abort an image transfer, or exit the program. The JPEG server transmits images using the JPEG File Interchange Format (JFIF [Ham92]). JFIF requires that all table specifications used in the encoding and decoding process (quantization and huffman tables) be available at both the client and server prior to their use.

The application is designed using results from [Heybey92] which describes the suitability of the ALF approach in video coding. Also parts of the XV program has been reused [Brad93]. The JPEG image is decomposed into a set of ADUs (Application Data Units) that can be received and processed out of order. However, the quantization and huffman tables must be transferred in order from the server to the client prior to the main image transfer. In other words, the level of reliability required for the tables is different to the rest of the image.

The application transmission control is composed of the following (see figure 5):

- The server first loads all the images into memory so that the file I/O is not included in the results of the experiments (1).
- The client asks for a specific image (2).
- The server sends all the JFIF table specifications (3) and then extracts ADUs from the image, compress them and sends them through the network (4) (5). For the performance evaluation, the extraction and compression (4) is done between (2) and (3) so that the troughput of the ADUs is not affected.
- The client receives ADUs that are decompressed and then displayed on the screen indepndently each from other (6) (7). For the performance evaluation, decompression (7) is not performed.

Three versions of this application have been developed:

- A hand-coded version that does not use the ALF concept (called No ALF),
- A hand-coded version based on ALF (called ALF), and
- An automatically generated version based on ALF (called ESTEREL) using the Protocol Compiler.

Two sets of experiments are described below. Firstly, the hand-coded implementations (ALF versus No ALF) are compared in order to validate the concept of ALF. The second experiment analyzes the efficiency of the automated approach. The throughputs represent the amount of compressed image data divided by the time spent to transmit it. Experiments were carried out on a lightly loaded live Ethernet (labelled Local Ethernet on next tables) and between France (INRIA) and Australia (UTS) (labelled Internet on next tables). Experiments were performed using Sun10 workstations running SunOS (the average value of 100 measures is given).

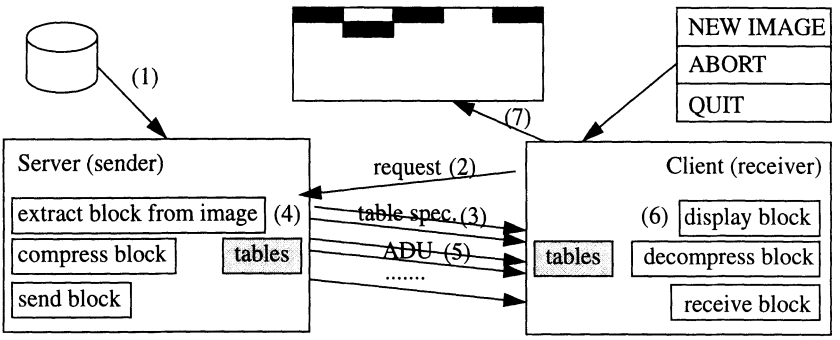

Figure 5: The JPEG player architecture.

4.1 Validation of the ALF concept

The two hand-coded implementations have been designed to investigate the effect of ALF on communication subsystems design and performance. The No ALF implementation runs over an in-kernel TCP. The ALF implementation runs over its own protocol which takes advantages of ALF. This protocol (called TPALF) is a user-level protocol that runs over UDP/IP. It was modified from the 4.3BSD TCP/IP implementation. The only changes were to allow out-of-order processing of incoming data and to handle ADUs as opposed to streams packets. Flow and congestion control is done with a sliding-window scheme using the slow-start algorithm. Error control is achieved through both Cumulative and Selective Negative Acknowledgments.

An issue was to determine the size of the unit of transmission. As the JPEG server ADUs are very short (about 60 bytes on average), several ADUs have been concatenated within one transmission unit. But ALF means also the ADUs must be preserved through the whole communication system and segmentation must be avoided. ALF is a strategy that organizes the transmitted packets into data meaningful to the application. This allows the receiver to process independently and immediately each packet received. When segmentation occurs within the protocol, the received packets cannot be delivered to the application on arrival, and the benefits of ALF would be lost. This is illustrated on table 1 (with a MTU size of 1460 bytes). Thus the size of the transmission unit should not exceed the size of the minimum MTU (Maximum Transmission Unit) of the network.

Table 1: Throughput with different packet sizes via Internet

	ALF	No ALF
512 bytes	14.7 Kbits/s	6.3 Kbits/s
1460 bytes	35.0 Kbits/s	8.3 Kbits/s
2000 bytes	7.2 Kbits/s	8.2 Kbits/s

The two hand-coded implementations have been also used to investigate whether applications can benefits from out-of-order processing. The comparison between table 1 and table 2 demonstrates how the non-ordered delivering gives the possibility of exploiting the internal parallelism of the application.

Table 2: Throuhput via Local Ethernet

	ALF	NoALF
1460 bytes	7.35 Mbits/s	7.67Mbits/s

The experiments through local networks (table 2) show that the No ALF implementation performs slightly better when the underlying network is reliable (almost no out-of-sequence data transmission). Through Internet (table 1), where delays and loss can produce out-of-order data delivery, ALF appears to be more efficient (300 % faster in the best case, where the receiver can processes immediately the ADUs when they arrive, whatever the order is). It shows that ALF improves the efficiency of the communication sub-system by handling the ordering at the application level. Further details are given in [Chrisment 94b].

4.2 Evaluating the automated approach

4.2.1 JPEG player specification

The automated implementation was created by using ESTEREL. Figure 6 shows the main part of the input application specification for the server process (only the declarations and exception handling have been omitted). The server waits for a remote image request (label 2 on figure 6). The first *present* statement specifies a reliable and ordered transmission of the table specifications (labels 3 on figure 6). The server then loops until all the ADUs are sent (labels 4 and 5). The second *present* statement does not request ordering which implies that the transmission (and reception) can be out of order within the block delimitated by the *present* instruction. However, as just implied by the ESTEREL syntax, order is guaranteed between the two *present* blocks. The specification of the client has a similar structure and length.

It can be seen that the specification uses ALF as firstly it has references to the user data structures, which are then passed directly to the communication protocol. Secondly, through the *present* construct, the protocol gives different reliability guarantees to different parts of the data transfer, allowing, when it is possible, out-of-sequence processing of ADUs.

```
do                                                          %watching User_Quit
   loop                                                     %normal image request
      var User_Context: CONTEXT in
         do                                                 %watching Remote_Abort_Req
            do                                              %watching Local_Abort
2           await Remote_ImagReq;
               var Last: boolean, ADU_To_Send: ADU in
                  User_Context := User_OpenCtx(?Remote_ImagReq);
                  present [Reliable and Ordered] do
                     call get_ADU_from_image(ADU_To_Send, Last)(User_Context, SPECQUANT);
3                    emit Remote_ADU(ADU_To_Send);
                     call get_ADU_from_image(ADU_To_Send, Last)(User_Context, SPECHUFF);
3                    emit Remote_ADU(ADU_To_Send);
                  end;
                  present [Reliable and FlowControl] do
                     trap End_Of_Image in                   %transmission of image ADUs
4                       call get_ADU_from_image (ADU_To_Send, Last)(User_Context, MCU);
5                       emit Remote_ADU(ADU_To_Send);
                        if Last then
                           exit End_Of_Image
                        end
                     handle End_Of_Image do
                        nothing
                     end trap
                  end;
               end var
   .....
   % catch the exceptions %
   .....
end
```

Figure 6: the application specification for the JPEG Server in ESTEREL. The label numbers correspond with those used figure 5.

The implementation is generated in two steps. Firstly the application specification of the JPEG server (figure 6) is translated into its integrated specification using the parser described in section 3.2. In the integrated specification, a dedicated user level transmission control protocol has been added to the application specification. The keywords within the *present* statements of the application specification are translated into ESTEREL templates and into standard protocol function calls. For example *FlowControl* is translated into a function call to FlowControl within the ESTEREL template used for *Reliable* transmission. This template introduces timers and specifies responses to timeout signals. The size of the server's specification changed from 74 to 248 lines after parsing, while the client's specification changed from 94 to 240 lines of ESTEREL code. On the automaton aspect, a 2 states automaton is produced for the server specification, and 5 states for the client specification. After protocol integration, both server and client are made of 5 states.

The second step in the compilation is that the intermediate ESTEREL specification is processed by the ESTEREL compiler to produce C code. This C code accesses C modules that describe the application and its data structures. These modules were written by the application programmer. The protocol function calls inserted by the parser are accessed from a protocol

library that is currently being developed. Finally the executable for the JPEG server was created by using a C compiler to compile and link all the modules together. The ESTEREL version of the JPEG player runs directly over UDP/IP.

4.2.2 Discussion

Three issues concerning the automated JPEG player implementation can be analyzed:

The design issue
The automated approach improves the flexibility and modularity of ALF based programs. This is best illustrated by an example. In the hand-coded implementation, the TPALF protocol is used during all the life of the application. The TPALF protocol allows out of order delivery of data. This is beneficial during the actual image transfer but goes contrary to design modularity, as the application must understand the protocol and explicitely reorder the JFIF table specifications. The protocol specified by ESTEREL changes during the life of the application, providing ordered delivery for the table specifications, and allowing out of order delivery of the image ADUs. Thus the application software was not required to implement any communication protocol functionality resulting in an improved software structure. The generalization of this result is that if a protocol is not able to re-configure its functionality during the life of a connection (using ESTEREL or by any other means) then it should be designed to provide a level of functionality somewhere between the highest and lowest levels required during the life of a connection. If the protocol always provides the highest level of functionality, then the benefits of removing protocol functions cannot be exploited when the extra functionality is not required (for example forcing data to be ordered when the application no longer requires it). Always providing a lower level of functionality implies that the application is responsible for the additional protocol functions when required (for example the application must re-order some of the data). This results in poorly structured software.

The performance issue
We compared the C code generated by the ESTEREL compiler with the ALF hand-coded implementation. The protocol specified in ESTEREL did not implement the slow-start algorithm but used a simpler flow control: the acknowledgement packets are generated after each 4th ADU and the window size has been fixed to 8. In order to obtain a fair performance comparison, we modified the initial TPALF protocol (see section 4.1) so that the both implementations use the same flow control parameters.

The results of this comparison are given in the table 3 and show that the choice of a formal and automated approach does not imply bad performances. Even, we observe that the ESTEREL automated implementation has a higher performance (20 %) than the hand-coded implementation over a unreliable network like Internet. The better reactivity of the automated code, due to the optimized automaton produced by the ESTEREL compiler, permits to improve the out-of-order processing and to better benefit of the ALF concept.

Over a more reliable network (like Ethernet), the ESTEREL automated implementation remains still better. The ESTEREL specification allows a better integration of the transmission control into the application. In the resulting automated code all protocol interfaces are suppressed except the user-kernel interface.

We also note that, over Ethernet, the network is the performance bottleneck. Further experiments over large bandwith networks (FDDI/ATM/Ethernet 100 base T) are forseen to analyze the performances when the bottleneck is due to the protocol processing.

The code issue
ESTEREL produces a code which is equivalent to the code written in C language. The executable code of the receiver side is even smaller (4% on 200.000 bytes). This can be explained because, in the ALF hand-coded version, the TPALF protocol is implemented as a user-level library contaning all functions even those never used either by the client or the server side. The automated approach allows to keep a certain level of modularity while producing a protocol more adapted and integrated to the application.

Table 3: Throughput between handcoded and ESTEREL versions

	local Ethernet	Internet
ALF	7.39 Mbits/s	51.3 Kbits/s
ESTEREL	7.42Mbits/s	62.1 Kbit/s

5.0 RELATED WORKS

Other research groups are currently working on the automated design and implementation of communication subsystems tailored to application requirements [Oeschlin94] [Plagemann92] [Schmidt93] [Richards94] [Diaz94] [Omalley 94]. The proposed solutions include developing general purpose protocols that allow flexibility. However these solutions are not operating system independent because the implementations are either part of the kernel, or a server within a micro-kernel based operating system.

Da CaPo [Plagemann92] is a more advanced tool for dynamic configuration of end-to-end transmission control protocols tailored to the application characteristics. A complex heuristic is used to design an independent control automaton for the end-to-end transmission control protocol (called CoRA). There is no integration, and the 3 layer architecture is respected. ALF is not retained as a design principle, which makes more complex the design of the communication support (itself implemented in the kernel space of the host computer). Da CaPo proves that tailoring protocol to the application characteristics is efficient in case of multimedia applications. It also proves that automated design in a formal framework is feasible with high performance.

[Diaz 94] also proposes a layered system where classic protocols are used to transmit appliction data, and where the application automaton is used to synchronize the data reveived (or transmitted) on the various protocols. The concept of "partial order connections" which is used to optimize the transmission, is very close from ALF. [Diaz 94] uses classic transport protocol; in the case of the JPEG player, it could have used TCP to transmit the control tables,

and UDP for the MCU packets. The application automaton is described using Timed Petri Nets. Protocols used, as well as the application automaton, are implemented in the kernel space of the host computer. A common experiment is being carried out to compare the two different approaches.

6.0 CONCLUSION AND FUTURE WORKS

It has been demonstrated that the automated integration of transmission control functions in an application formal specification is possible both in theory and in practice. Performance results confirm that, in term of code organization, size, and efficiency, the automated approach is almost as performant as the hand-coded approach. Using this approach, a completely automated implementation of distributed applications is credible.

The present parser design at present is limited to the integration of the control description of the transmission control to the application specification. It is being extended with:

- A richer heuristic to identify the most efficient transmission control facilities to be integrated. This is linked to the definition of a new QoS model adapted to the formal and automated approach.
- Dedicated verification tools. In a client/server model, some verifications have to be done on the coherency between the pair entities. This is made easier by the use of the formal framework since the begining of the design.
- An optimized implementation generator. The ILP criteria (Integrating Layer Processing [Clark90]) of optimization (instead of the one used in the current ESTEREL code generator) will be designed. ILP results in more performant implementation of the data manipulation functions, minimizing read/write operation which are known to be very costly.

The targeted system will be a new generation RPC-like generator, dedicated to multimedia applications. This RPC-like generator will, starting from the ESTEREL specification of a distributed application, integrate automatically the communication support required for optimal operation, and then generate the client and the server stubs. Such a tool will be useful to investigate new communication architecture (prototyping various multimedia applications with different QoS requirements); but also to design tailored communication facilities for multimedia applications where the complexity of flow synchronization does not allow an efficient hand-coded design.

7.0 ACKNOWLEDGMENT

Antony Richards gratefully acknowledges the support from Telecom Australia.

8.0 REFERENCES

[Berry92] G. Berry and G. Gonthier. "The Esterel Synchronous Programming Language: Design, Semantics, Implementation". Journal of Science Of Computer Programming. Vol. 19, Num. 2, pp. 87-152. 1992.

[Bradley93] J. Bradley. "XV Interactive Image Display for the XWindow System Version 3.00". April 1993.

[Castel94] C. Castelluccia and W. Dabbous. "Modular Communication Subsystem Implementation using a Synchronous Approach". Usenix Symposium on High Speed Networking. Oakland, August 1994.

[Chrisment94a] I. Chrisment and C. Huitema. "Remote Operation System Tailored to Application Requirements". IFIP International Conference ULPAA '94. Barcelone. June 1994.

[Chrisment94b] I. Chrisment. "Impact of ALF on Communication Subsystems Design and Performance". Proceedings of the First International Workshop on High Performance Protocol Architectures 94. December 1994. Sophia Antipolis.

[Crowcroft92] J. Crowcroft, I. Wakeman and Z. Wang. Layering Considered Harmful. IEEE Network, Vol 6, N.1, January 92

[Clark90] D. D. Clark, D. L. Tennehouse. Architectural Considerations for a New Generation of Protocols. Proceedings of ACM SIGCOMM. 1990.

[Corbin91] J. R. Corbin. "The Art of Distributed Applications". SUN technical Reference Library. Springer-Verlag Editor. 1991.

[Diaz94] M. Diaz, C. Chassot, and A. Lozes. "From the Partial Order Connection Concept to Partial Order Multimedia Connections". First HIPPARCH workshop. INRIA Sophia Antipolis. December 15-16, 1994.

[Diot94] C. Diot, "Communication Protocol Development using ESTEREL". First International HIPPARCH workshop. INRIA Sophia Antipolis. December 15-16, 1994.

[Diot95] C.Diot, C. Huitema, and T. Turletti. "Network Conscious Applications". IEEE Workshop on High Performace Communication Systems. Mystic. August 23-25, 1995..

[Ham92] E. Hamilton. "JPEG File Interchange Format Version 1.02". C-Cube Microsystems. September 1992

[Heybey92] A. T. Heybey. "Video Coding and the Application Level Framing Protocol Architecture". MIT report MIT/LCS/TR-542. June 1992.

[Hoare79] C. A. R. Hoare, "Communicating Sequential Process", Communication of the ACM, April 1979.

[Hoglander94] A. Hoglander. "Experimental Evaluation of TCP in User Space". INRIA Internal Report. September 1994.

[Meije94] "Tk Meije environment MAN pages". WWW server http://zenon.inria.fr:8003/meije/meijetools.html. Sophia Antipolis. 1994.

[Huffman62] D.A. Huffman. "A method for the construction of minimum redundancy codes". Proceedings IRE. Vol 40. pp. 1098-1101. 1962.

[ISO87b] ISO/OSI "Specification of Abstract Syntax Notation One (ASN 1)", Geneva, July 1987.

[Nelson81] B. J. Nelson. "Remote Procedure Calls". ACM Transactions on Computer Systems. May 1981. (Also PhD thesis, CMU-CS-81-119).

[Oeschlin94] P. Oeschlin and S. Leue. "Enhancing ILP using Common Case Anticipation and Data Dependence Analysis. First HIPPARCH workshop. INRIA Sophia Antipolis. December 15-16, 1994.

[Omalley94] S. W. O'Malley, T. Proebsting, and A. B. Montz. "USC : A Universal Stub Compiler. Proceedings of ACM '94. Vol. 24, N. 4. October 1994.

[Plagemann92] T. Plagemann, B. Plattner, M. Vogt, T. Walter. "A Model for Dynamic Configuration of Light-Weight Protocols" Proceedings of the third workshop on FTDCS. Tapei. Taiwan. pp. 100-110. April 1992.

[Richards94]A. Richards, A. Seneviratne, M. Fry and V. Witana. "Tailoring the Transport Protocol for Giga Bit Networks". In the Australian Telecommunication Networks and Applications Conference. 5-7 December 1994. ftp://ftp.ee.uts.edu.au/pub/prose/richards.atnac94.ps.gz

[Schmidt93] D. Schmidt, B. Stiller, T. Suda, A.N. Tantawy, and M. Zitterbart."Language Support for Flexible, Application-Tailored Protocol Configuration". Proceedings of 18th conference on Local Computer Network. 1993.

[Wallace90] G. Wallace. "Overview of the JPEG (ISO/CCITT) still image compression standard. Image Processing Algorithms and Techniques". In Proceedings of SPIE. Vol. 1244. pp. 220-233. February 1990.