REIHE INFORMATIK
5/94

Implementing Movie Control, Access and Mangement – from a Formal Description to a Working Multimedia System

R. Keller, S. Fischer und W. Effelsberg
Universität Mannheim
Seminargebäude A5
D-68131 Mannheim

Erschienen in:
Proceedings of the 14th International Conference on Distributed Computing Systems,
Poznan, Poland, June 1994

Implementing Movie Control, Access and Management – from a Formal Description to a Working Multimedia System

Ralf Keller, Stefan Fischer and Wolfgang Effelsberg
University of Mannheim, Praktische Informatik IV
D-68131 Mannheim, Germany

Abstract

In this paper we describe the tool–supported specification and implementation of a multimedia communication protocol on parallel hardware. MCAM is an application layer protocol for movie control, access and management. We specify the full MCAM protocol together with ISO presentation and session layers in Estelle. Using a code generator, we derive parallel C++ *code from the specification. The code is compiled and executed on a multiprocessor system under OSF/1 and on UNIX workstations. Measurements show the performance speedup gained by several different configurations of parallel units. We also report on experiences with our methodology.*

Keywords: *multimedia systems, application layer, formal specification, parallelization, code generation*

1 Introduction

The main purpose of this work is to prove that it is possible to specify and implement a new application layer protocol using formal description techniques and code generators; moreover that it is possible to derive efficient implementations for a distributed heterogeneous and even parallel environment. Our method works on a wide range of platforms, from single–processor workstations from different vendors (DEC, IBM, Sun) to multiprocessor parallel computers such as a KSR1 (Kendall Square Research [8]). The inclusion of a multiprocessor system enables us to increase the performance of our system by exploiting internal parallelism in protocols and by handling several connections in parallel.

There are two factors propelling our work. The first one being that communication software has become the major bottleneck in high–speed networks, especially for the upper layers of the protocol stack [3]. The second involves the hard requirements in terms Quality of Service guarantees which must be fulfilled in distributed multimedia systems. Take, for example, a digital video–on–demand service. Current high–end workstations can serve only a small number of clients simultaneously. But imagine systems in which one machine has to serve thousands of clients simultaneously without noticeable performance degradation as customers would be paying for the service and expecting a good quality. We investigate the use of parallel computers for this purpose.

It can be expected that the performance difference between high–speed transmission systems and communication software will not shrink (rather, will probably even increase) within the next few years. Several lightweight protocols for faster computation have been designed and implemented [4]; but the speedup achievable with this method is limited. Also implementation code was fine–tuned by experts to improve performance [18]; our impression is that existing protocols are so well understood now that further improvements will be very difficult. A much greater speedup can be gained by using several processors to do the same work. Earlier work on the parallel execution of lower layers has resulted in a promising speedup [1, 26].

This paper is organized as follows. In Section 2, we present the architecture of MCAM, our example protocol. Section 3 describes our experimental setup. Our implementation methodology is the subject of Section 4. In Section 5, we present preliminary performance measurement results gained by the use of parallelism. We also report on experiences with our methodology. Section 6 concludes the paper.

2 MCAM Architecture

MCAM is an application layer architecture, service and protocol for Movie Control, Access, and Management in a computer network [19]. Our architecture allows a user to *access* (create, delete and select), *manage* (query and modify attributes), and *control* (playback or record) movies.

	control protocol	**CM stream protocol**
data rates	low	high
reliability	100%	$\leq$ 100%
error correction	yes	lightweight or none
timing relations	asynchronous	isochronous
delay and jitter control	no	yes
$\Rightarrow$ protocol stack	**OSI** or **TCP/IP**	**XMovie/MTP**

Table 1: Different requirements of the protocol types

From ongoing work on our XMovie project [21] we have gained experience in implementing CM–streams (continuous media streams, e.g. video) in a network. We have learned from XMovie and other projects that while low–level stream services can be implemented on today's computers successfully, they are currently limited by severe bottlenecks [20]. For the realization of a practical multimedia service in a distributed environment, two additional support services are absolutely necessary: movie directory and CM equipment control.

The movie directory is used as a repository for movie information, such as digital image format and storage location. The equipment control service enables the user to control CM equipment attached to remote computer systems, e.g. speakers, cameras, and microphones.

In contrast to other well known architectures such as the Lancaster architecture and Bellcore's Touring Machine [24], we separate the control protocol from the CM stream protocol incorporating the two new support services into the control protocol. This separation better accommodates the varying requirements for stream protocols for continuous media streams and control protocols in terms of data rate, reliability, error correction, timing relations, delay and delay jitter control (see Tab.1).

Thus we propose to run the two application protocols over different protocol stacks. The control protocol needs a reliable service but generates low data rates only; we have decided to use the OSI protocol stack for the control protocol.

In contrast, a CM stream protocol has to be placed on an isochronous (possibly unreliable) stream service which provides high data rates; we have chosen the XMovie stream service developed at the University of Mannheim.

The functional model of our MCAM architecture is shown in Fig 1. It consists of four parts: Directory System, Equipment Control System (ECS), Stream Provider System (SPS), and MCAM (see [19] for more details).

3 Experimental Environment

The experimental setup for our performance analysis is shown in Fig. 2. MCAM clients on different systems control CM–streams sent by MCAM server entities. All these server entities can run simultaneously on a multiprocessor system.

Our system places the MCAM control protocol on two different OSI protocol stacks, thereby allowing us to test conformance, and to compare run–time performance. The first one generates the presentation and session layers out of an Estelle specification which can run in parallel on top of the ISODE transport layer. The second stack places the MCAM module directly on top of the ISODE presentation interface. With these two versions we can measure performance differences between generated and hand–written code.

In addition we can compare serial and parallel implementations of our protocol stack by distributing Estelle modules over a set of parallel processors on the KSR1 machine. Initial experiments have shown that connection–per–processor will yield better performance than layer–per–processor [6]. The actual speedup depends largely on the mapping of modules to the tasks and threads of the operating system as described in the next section.

In addition to the two complete control protocol stacks for MCAM, we have implemented the CM protocol stack. With a standardized real–time transport protocol still lacking, we run the XMovie transmission protocol MTP directly on top of UDP, IP and FDDI. The experimental protocol stacks described above are realized in the following hardware/software environment: The client entities are implemented on one–processor UNIX workstations (Sun and DEC), while the server machine is a 32 processor KSR1 running OSF/1 [22]. All the software is in C++, and we are using ISODE v8.0.

4 Implementation Methodology

Before implementing a protocol we first specify it formally in a standardized formal description language. Formal description techniques improve the correctness of specifications by avoiding ambiguities and

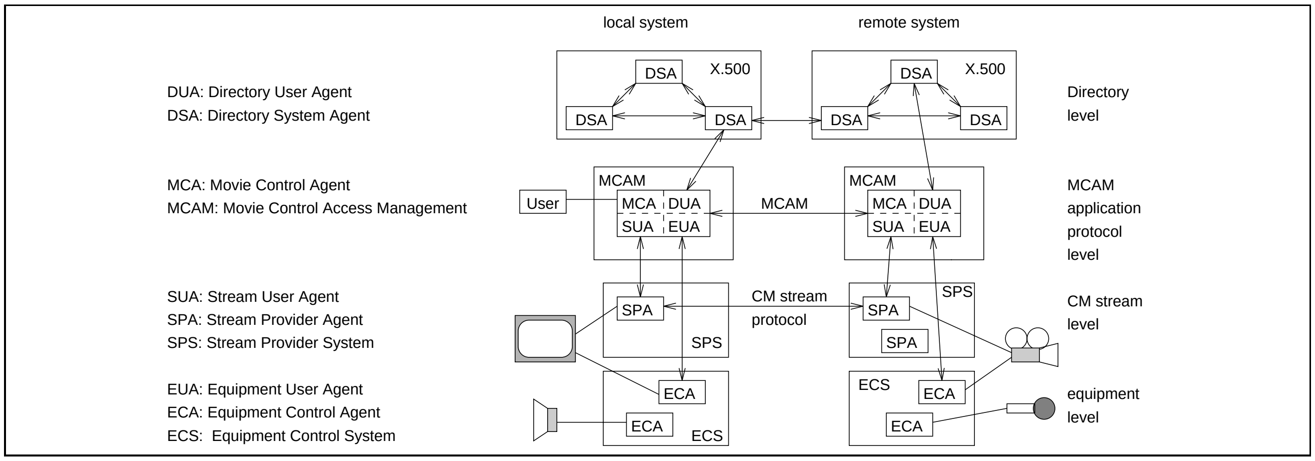
local system
remote system
DUA: Directory User Agent
DSA: Directory System Agent
DSA
X.500
DSA
DSA
DSA
X.500
DSA
DSA
Directory level
MCA: Movie Control Agent
MCAM: Movie Control Access Management
User
MCAM
MCA
DUA
SUA
EUA
MCAM
MCAM
MCA
DUA
SUA
EUA
MCAM application protocol level
SUA: Stream User Agent
SPA: Stream Provider Agent
SPS: Stream Provider System
SPA
SPS
CM stream protocol
SPA
SPS
SPA
CM stream level
EUA: Equipment User Agent
ECA: Equipment Control Agent
ECS: Equipment Control System
ECA
ECA
ECS
ECS
ECA
ECA
equipment level

Figure 1: MCAM functional model

by enabling formal verification. Tools can be used to verify the correctness of the specification. In addition, formal specifications allow semiautomatic code generation. Generating code from specifications has several advantages: The code can be maintained more easily since the system is specified in an abstract, problem–oriented language. The implementation code has fewer bugs and is well–structured. It is also much easier to port an implementation to another system.

We use the formal description language Estelle [17] which is mainly based on finite state machines (FSMs). An Estelle specification consists of a hierarchically ordered set of FSMs called *modules* communicating via bidirectional links called *channels*. Estelle modules can be nested: Within the body of a module, other modules, called *child modules*, can be defined. Thus all modules of a specification form a tree.

The execution sequence of the modules is controlled in two ways: either according to their position within the tree, or by means of an attribute given to each module. The basic tree rule is that a parent module always takes precedence over its children, i.e. a child can only execute if the parent has nothing to do. A parent and a child can never run in parallel.

The *module attributes* control the parallelism between modules at the same level of the hierarchy. There are four attributes: `systemprocess`, `systemactivity`, `process` and `activity`. In the following we use the term *system module* for systemprocesses and systemactivities. Then the following Estelle rules apply:

- Every active module must have one of the four attributes.
- A `system` module cannot be contained in another attributed module.
- Each `process` module and each activity module must be contained (perhaps indirectly) in a system module.
- A `process` module or a `systemprocess` module can contain other `process` or `activity` modules.
- An `activity` module or a `systemactivity` module can only contain other `activity` modules.
- Children whose parent module is of type `process` or `systemprocess` may all run in parallel. Children whose parent module is of type `activity` or `systemactivity` are mutually exclusive, i.e. only one of them can run at a time.

As a consequence, a module containing a `system` module must be inactive; it is typically located at the root of a tree. Also, in each path of the tree, from the root to a leaf, there is exactly one system module, i.e. each active module belongs to exactly one system module.

The dynamics are as follows. At runtime, a module instance can only be dynamically created and destroyed by its parent module. Thus the number of module instances can vary at runtime, but their relative position within the tree is predetermined. When the system is initialized, exactly one instance of each `system` module is created. As opposed to the `activity` modules and the `process` modules, the structure of the `systemactivity` modules and `systemprocess` modules is static at runtime. The

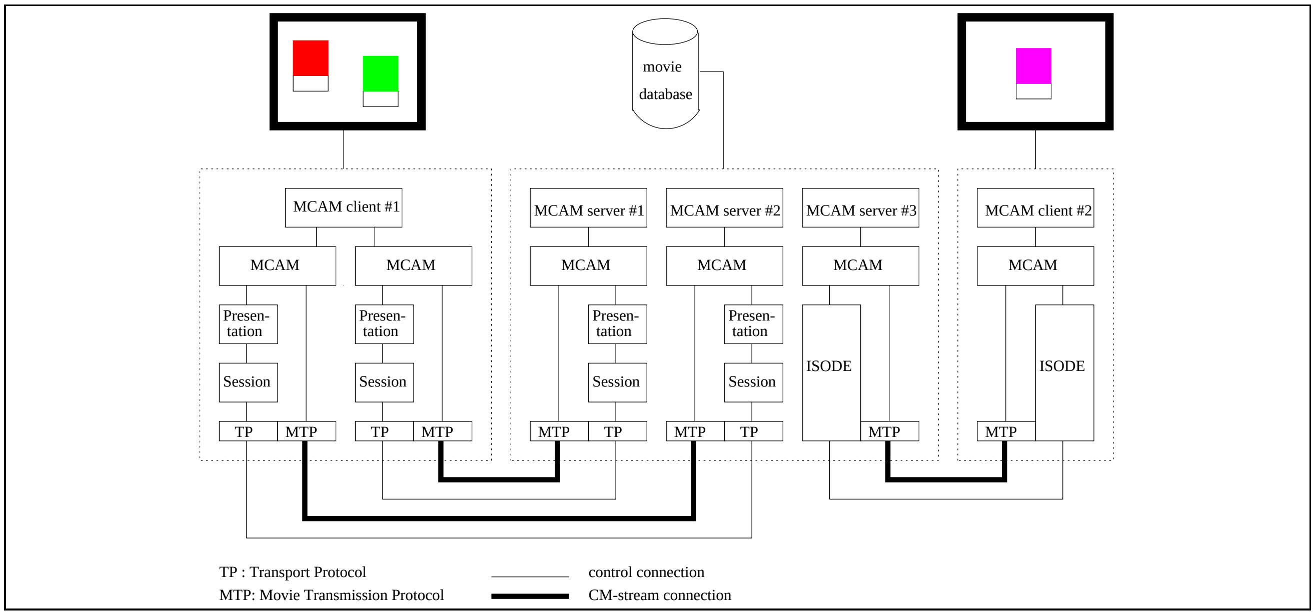

Figure 2: An example configuration

`system` modules themselves are mutually independent and can run asynchronously and in parallel.

The motivation behind these semantics is that a typical communication system has static parts and dynamic parts (to be created at runtime). For example a protocol entity implemented as a `process` can accept a new CONNECT request and then create a new child module to handle the new connection. All child module instances for parallel connections will then be independent of each other and able to execute in parallel.

The way to an implementation of MCAM is a four–step process:

1. Specification in Estelle
2. Code Generation (C++)
3. Inclusion of hand-coded parts
4. Compilation of C++ modules

Each of these steps is described in detail in the following subsections.

4.1 Specification and Initialization

Each MCAM instance consists of four modules: Movie Control Agent (MCA), Directory User Agent (DUA), Stream User Agent (SUA) and Equipment User (EUA). Therefore the mapping of MCAM modules to Estelle modules is straightforward (see Fig. 3). Only the MCA module is completely written in Estelle (header and body), whereas the three remaining ones describe only their interface in Estelle with their module body written in C or C++. So we can very easily access existing services such as the movie directory out of our Estelle specification.

The application interface shown in Fig. 3 provides a set of procedures to the MCAM user. The presentation interface connects the MCAM specification to the lower layers, which are in our case provided by ISODE, the ISO Development Environment [23] or alternatively by Estelle implementations of ISO presentation and session layers.

The whole system is specified as follows: for the server and for each client, we generate an Estelle `systemprocess` module. In comments, we declare the location (i.e. a machine name) where the module will be placed in the implementation. Such modules may run in parallel, following the Estelle semantics. However, the number of `systemprocess` modules cannot be changed at runtime, so the number of clients is fixed. Each client can open several connections to the server, but it is not possible to dynamically generate new clients on other machines.[1] This disadvantage is compensated by the flat structure of the specifica-

[1]An Estelle enhancement enabling dynamic generation of clients is described in [2].

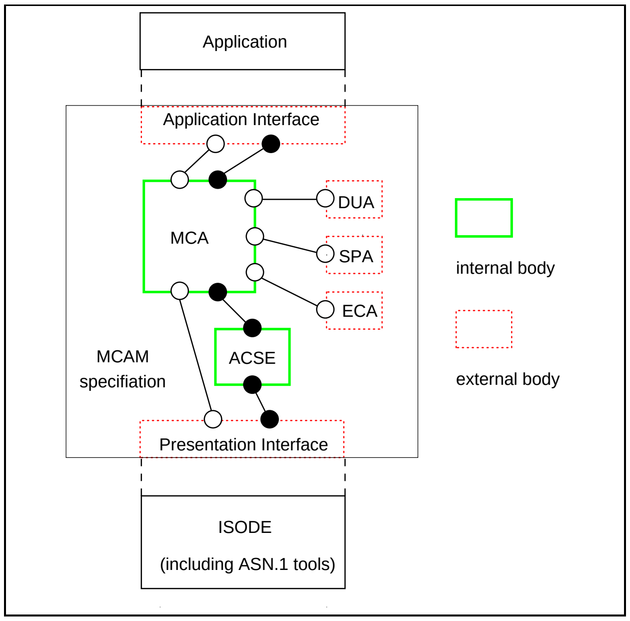

Figure 3: Mapping to Estelle modules

tion, making the implementation faster. Client and server modules have the ability to create submodules, which implement the protocol. Each client module creates an application module, which allows a user to send requests to and receive answers from the MCAM system. When a connection request is received from the application module, a client module will create an MCAM module and either presentation and session modules[2] or an ISODE interface module. The connection request will be transmitted to the server, who then creates the same Estelle modules. The new modules are connected by an Estelle channel. In the case of Estelle presentation and session, the corresponding session modules are connected to each other, while in the ISODE, the two interface modules are connected. Then, the MCAM connection is set up, and the client application may start its communication with the new server entity.

4.2 Code Generation

As already mentioned, it is possible to semi–automatically generate code from formal specifications. However, one of the major problems of generated code has been performance. Existing code generators were made to easily get rapid prototypes from Estelle specifications; the goal of these generators was to create an executable version of the specification for validation purposes, rather than to produce efficient runtime code. Use of parallelism was limited to validation and simulation purposes. We implemented a modified code generator based on the Pet/Dingo system [25] that reads an Estelle specification and produces C++ code for parallel programs running under OSF/1 [6]. It maps Estelle modules onto OSF/1 parallel processes, i.e. tasks and threads (lightweight processes). In its current version, the generator's runtime system produces one thread for each Estelle module, creating the maximum degree of parallelism allowed by Estelle semantics.

On a single–processor machine there will obviously be no speedup. This is true for the two clients in our experiment. The server, however, runs on the KSR1 machine, and makes use of two forms of parallelism: first, all the layers run in parallel, i.e. the application layer protocol MCAM, and presentation and session layer, resp. ISODE. Second, we have a per–connection parallelism. A discussion of these and other forms of parallelization of protocols can be found in [15].

The code for the body of the ISODE interface module cannot be generated, as the module is not specified in Estelle. In the specification, the module body is declared `external` which instructs the compiler to generate only the C++ interface but not the implementation. The latter has to be added by hand, and is described in section 4.3.

The body of the application module cannot be expressed in Estelle either, as it should present an X interface to the user. However, we implemented a tool [10, 13] which creates an X interface accepting the description of the channel between application and MCAM module as an input. Any message sent by the application can be invoked via a button–click by the user; a window is then opened on the screen where the user may specify details of the message. Incoming messages are displayed in a window at the time of their arrival.

All MCAM PDUs are specified in ASN.1, an ISO defined data description language [16]. This ASN.1 specification is used to generate C++ data structures and to create encoding and decoding routines for our implementation automatically. In order to be able to use the ASN.1 specifications for the MCAM PDUs within Estelle, we had to implement another translator [9].

[2]The Estelle sources for the presentation and session layers were provided by the University of Bern, Switzerland.

4.3 Inclusion of Hand-coded Parts

The task of the ISODE interface module is to get messages from the Estelle interaction point and map them onto ISODE library calls like `PConnect-request()`. Also, it has to look for incoming ISODE messages, map them onto Estelle interactions and output them on the Estelle interaction point. This process is described in more detail in [5]. In principle, the execution loop of the ISODE interface module has the following structure:

```
while true do
  if (IP.message) then
    encode message in ISODE param. format
    call appropriate ISODE function
  endif
  if (ISODE.message) then
    encode message in Estelle param. format
    output IP.message
  end
end
```

4.4 Compilation

The code generated from Estelle and the hand–coded parts of the software are compiled using a C++ compiler and linked together. For each `system-process` module and for the specification root module, we create an executable file. It is necessary to build these files on each target machine, i.e. the specification and the server executable are built on the KSR1, while the client executable is built on all machines to be used as clients.

The specification module is started by hand on the server machine. It will then start the server itself and the specified number of clients on the different client machines. The information on where to start a client is taken from the comments in the Estelle source. Each client will start an application module that presents an X window interface to the user. The user can then send requests to his client module, e.g. to open and playback movies.

5 Results and Experiences

The main goal of this project is to fulfill the QoS requirements of multimedia applications. Thus, we have to achieve performance gains by using parallelism and minimize the difference in performance results between hand-coded and generated software. At the moment, the environment shown in Fig. 2 is completely implemented with the exception of the MCAM module which is in the debugging phase. We already have some measurement results on the influence of parallelism for the session and presentation layers.

5.1 Sequential vs. Parallel Implementation

For our first measurements we specified a simple test environment in Estelle with two protocol stacks connected by a simulated transport layer pipe. Both stacks consist of presentation and session layers, and an initiator or responder respectively. It is possible to create multiple connections. For the tests, we used presentation and session kernel, without ASN.1 encoding/decoding[3], and we transmitted very small P–Data units. This is the worst case for parallelization. Even with this environment, we got a speedup (in comparison with the sequential version) of 1.4 to 2 with 2 connections, parallel presentation and session and a varying number of Data requests. Higher performance gains can be reached with full protocols[14].

5.2 Influence of Mapping Alternatives

In the current version of the code generator, we map each Estelle module onto a lightweight process in OSF/1, allowing the maximum degree of parallelism. However, this is not always the best alternative. Consider the situation in which the number of Estelle modules exceeds the number of processors. Then, some modules have to share a processor. We then have not only the synchronization overhead between the threads running the modules, but also gain nothing from the parallelization. Our solution to this problem is to group certain Estelle modules into one unit, and run this unit by one thread. We take as many of these units as there are processors. There will still be no performance gains by parallelism, as all the modules in one group are executed sequentially, but this will avoid synchronization losses. First measurements with the new grouping scheme show further performance gains. Details can be found in [7].

In addition, it may be useful to further parallelize an existing specification. Modules which perform several long–running computations sequentially may be split in two or more modules resulting in a module pipeline where data is processed in parallel. The right decision of whether to integrate modules or split them depends highly on the module runtime and on the

[3]One might expect performance gains for parallel encoding/decoding. In [12], we show that by parallelization in this area, we do not obtain better performance.

performance requirements of the user. For protocols with only small processing times, the only useful parallelization will be the mapping of one connection to one processor, as those modules will not exchange data and thus need no synchronization.

Another important point for modules with large transition lists is the mapping of transitions. Mainly, there are two alternatives: first, each transition may be hard-coded as a C++ code block in a transition selection function. Prioritized transitions will have their place at the beginning of the function. Second, states and transitions may be mapped to a table. The current state will be used as an index for the row which means that only the enabled transitions for that state will be investigated. As newer performance measurements show, the table–controlled approach is significantly better than the hard-coded one [11] when the number of transitions becomes larger than four.

For protocols with small processing time, the Estelle scheduler of many available compilers becomes the bottleneck for the speedup. Measurements show a runtime percentage of the scheduler of up to 80%. Our scheduler shows better runtime behavior, as it is decentralized. Each part only has to check the transition of one module. This can be done in parallel.

5.3 Experiences using Estelle

In section 4, we already mentioned some of the advantages of using formal languages like Estelle in the protocol and system specification process. From this and previous work, we were able to make some experiences with the usage of Estelle compared, e.g., to the usage of C++.

Estelle is very easy to learn, even for the beginner. From our experiences with students learning Estelle, we can say that the period of adjustment is much less than one month. This is mainly due to the well–known concept of communicating finite state machines. We consider it much more difficult to learn process algebra–based languages like LOTOS.

Specifying a whole system or single protocols can be done much faster in Estelle than in C++. The language provides all means for defining communication structures and data types. Data type definition in C++ is equally fast, but communication programming is much more work if done from scratch!

However, one serious drawback of Estelle specifications is their not so well understood integration into real environments. This includes network attachment as well as user integration. Therefore, test case generation and distributed execution of the specified system needed some more work. Out of this experience, we currently develop a methodology of how to automatically integrate Estelle specifications in arbitrary environments.

We conclude that the usage of Estelle as a specification language for distributed and parallel system is well suited, but there has still some work to be done to simplify real implementations.

6 Conclusion and Outlook

One of the major problems of Estelle in a real–time environment is that QoS parameters cannot be specified. We cannot describe a realtime protocol like the XMovie stream protocol in Estelle because we cannot specify the QoS requirements of such a protocol; no hints are given to the code generator as far as performance is concerned. Non–realtime protocols such as MCAM also have QoS requirements, e.g. maximum delay of an interaction, but these are not as critical to our implementation as the realtime requirements are to the stream protocol.

One of the first lessons we have learned during our work on this system is that the mapping of Estelle modules to tasks and threads influences the performance of the runtime implementation to a great extent. An algorithm for an optimal mapping is currently under development [7].

As a final goal we envision a distributed MCAM system that is highly portable and runs on a great number of single- and multiprocessor platforms in an ATM network, providing a digital movie service in a heterogeneous environment.

References

[1] M. Björkmann and P. Gunningberg. Locking effects in multiprocessor implementations of protocols. In *Proc. of ACM SIGCOMM 93*. ACM, Sept. 1993.

[2] J. Bredereke and R. Gotzhein. Increasing the concurrency in Estelle. In R. L. Tenney, P. D. Amer, and U. Uyar, editors, *Formal Description Techniques VI – Forte'93, Boston, USA*. Participants' proceedings, Oct. 1993.

[3] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. *Computer Communication Review*, 20(4):200–208, Sept. 1990.

[4] W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meister, H. Rudin, and R. Williamson. A Survey of Leight-Weight Transport Protocols for High-Speed Networks. *IEEE Transactions on Communications*, 38(11):2025–2039, Nov. 1990.

[5] T. Fellger. Konzeption und Implementierung einer Estelle–ISODE Schnittstelle. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1993. (in German).

[6] S. Fischer and B. Hofmann. An Estelle Compiler for Multiprocessor Platforms. In R. L. Tenney, P. D. Amer, and U. Uyar, editors, *Formal Description Techniques VI – Forte'93, Boston, USA*. Participants' proceedings, Oct. 1993.

[7] S. Fischer, B. Hofmann, and W. Effelsberg. Efficient Configuration of Protocol Software for Multiprocessors. Technical report, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1994.

[8] S. Frank, H. Burkhard, and J. Rothnie. The KSR1: High Performance and Ease of Programming, no longer an Oxymoron. In H.-W. Meuer, editor, *Supercomputer '93*, Informatik aktuell, pages 53–70. Springer Verlag, Heidelberg, 1993.

[9] T. Goebel. Konzeption und Implementierung eines ASN.1–C++ Übersetzers. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1993. (in German).

[10] A. Grössler. Eine Benutzerschnittstelle für Estelle–Spezifikationen. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1993. (in German).

[11] T. Held. *Practice of the Computer–aided Protocol Implementation*. PhD thesis, TU Magdeburg, Fakultät Informatik, 1993. (in German).

[12] S. Herbert. Entwurf und Implementierung eines parallelen ASN.1–Encoders und –Decoders. Master's thesis, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1991. (in German).

[13] C. Hölzen. Generierung einer Benutzerschnittstelle für Estelle–Spezifikationen. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1993. (in German).

[14] B. Hofmann. *Deriving Efficient Protocol Implementations from Estelle Specifications*. PhD thesis, Universität Mannheim, Praktische Informatik IV, 1994. (in German).

[15] B. Hofmann, W. Effelsberg, T. Held, and H. König. On the Parallel Implementation of OSI Protocols. In *Proc. HPSC'92*, Tuscon, Arizona, USA, Feb. 1992.

[16] Information processing systems – Open Systems Interconnection – ASN.1 and its Encoding Rules. International Standard ISO 8824/25, 1988.

[17] Information processing systems – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.

[18] V. Jacobsen. Efficient Protocol Implementation. In *ACM SIGCOMM'90 tutorial*, Philadelphia, PA, Sept. 1990.

[19] R. Keller and W. Effelsberg. MCAM: An Application Layer Protocol for Movie Control, Access, and Management. In *Proc. ACM Multimedia'93*, pages 21–29, Los Angeles, CA, Aug. 1993.

[20] R. Keller, W. Effelsberg, and B. Lamparter. Performance Bottlenecks in Digital Movie Systems. In *Proc. NOSSDAV'93*, pages 163–174, Lancaster, U.K., Nov. 1993.

[21] B. Lamparter and W. Effelsberg. X-MOVIE: Transmission and Presentation of Digital Movies under X. In *Proc. NOSSDAV'91*, LNCS 614, pages 328–339. Springer-Verlag, Heidelberg, 1992.

[22] A Guide to OSF/1: A technical Synopsis. O'Reilly & Associates, Inc., 1991.

[23] M. T. Rose. *The Open Book: A Practical Perspective on OSI*. Prentice Hall, Englewood Cliffs, 1990.

[24] L. Ruston, G. Blair, G. Coulson, and N. Davies. Integrating Computing and Telecommunications: A Tale of two Architectures. In *Proc. NOSSDAV'91*, LNCS 614, pages 57–68. Springer-Verlag, Heidelberg, 1992.

[25] R. Sijelmassi and B. Strausser. The PET and DINGO tools for deriving distributed implementations from Estelle. *Computer Networks and ISDN Systems*, 25(7):841–851, 1993.

[26] M. Zitterbart. *Funktionsbezogene Parallelität in transportorientierten Kommunikationsprotokollen*. PhD thesis, Universität Karlsruhe, 1990. (in German).