25
On the Influence of Semantic Constraints on the Code Generation from Estelle Specifications

Ralf Henke, Andreas Mitschele-Thiel
Universität Erlangen-Nürnberg, Lehrstuhl für Informatik VII, Martensstraße 3, 91058 Erlangen, Germany, Phone/Fax: +49 9131 85 7932 / 7409, email: mitsch@informatik.uni-erlangen.de

Hartmut König
Brandenburgische Technische Universität Cottbus, Institut für Informatik, Postfach 101344, 03013 Cottbus, Germany, Phone/ Fax: +49 355 692236, email: koenig@informatik.tu-cottbus.de

Abstract

Implementations automatically derived from formal descriptions often do not fulfill the performance requirements of real-life applications. There are several reasons for this. In the paper, we discuss for the FDT Estelle how semantical constraints can influence the efficiency of the generated code. In the first part of the paper we show that certain language features may have a restraining effect on the performance of the implementation. The second part of the paper investigates how the activity thread model, a technique known from manual protocol implementation, can be applied to automatically derive efficient implementations from Estelle specifications. The activity thread model reduces the communication overhead. It is known to be efficient as the server model usually applied. We analyze the prerequisites to apply the model and present measurements comparing the performance achievable with the technique. The measurements are given for different implementations of XTP and the XDT protocol.

Keywords

Estelle, semantics, automated protocol implementation, activity thread model.

1 MOTIVATION

The performance of communication systems is decisively determined by the protocols used. Experience with practical protocol implementations shows that protocol performance depends as much, and usually more, on the implementation than on the design [Wats87] [Clar89]. Communication protocols are usually implemented by

Formal Description Techniques and Protocol Specification, Testing and Verification
T. Mizuno, N. Shiratori, T. Higashino & A. Togashi (Eds.)

hand. The implementations are typically based on informal standards provided by ISO, ITU-T or other organizations. The techniques employed for hand-coded implementations have been continuously improved. Examples range from the simple elimination of copy operations between protocol layers by data referencing to more sophisticated optimizations as application level framing and integrated layer processing (ILP) [Clar90][Ahlg96]. In addition, the exploitation of various kinds of parallelism to derive efficient protocol implementations has been investigated, but it has not brought the expected efficiency gain.

Product implementations of communication protocols that are automatically derived from formal descriptions are less reported, although approaches for the computer-aided derivation of implementations have been investigated for more than ten years. So far, the use of automatically derived implementations is limited due to their lack of efficiency. Thus, they are mainly used for prototyping or for applications where optimal performance is not crucial.

The main obstacle for the successful application of computer-aided implementation techniques (and probably of formal description techniques in general) is the insufficient efficiency of the generated code. There are several reasons for this [Held95]:

(1) The implementation model prescribed by the transformation tool is elaborated during tool design. It does not take into account the context, in which the implementation is used.
(2) Formal description techniques are based on formal semantics. The semantics highly influence the design of the implementation model. Optimizations can only be introduced by ensuring correctness and conformance criteria.
(3) So far, automated protocol implementations are mainly focused on one-layer implementations (due to their orientation on prototyping).

In recent years the derivation of efficient implementations from formal descriptions has been investigated in several papers. They have brought interesting experimental results, especially for the FDTs based on extended finite state machines (EFSM) as Estelle and SDL. However, the question whether FDT-based implementations can compete with hand-coded ones in performance has not finally been answered yet. The approaches pursued in these papers were different. The influence of specification styles on the runtime efficiency is among others discussed in [Gotz96]. In [Held95] experiments with an experimental Estelle-C compiler are reported that applies optimized algorithms for implementing certain features of the Estelle semantics, e.g. for the synchronisation of module instances for the selection of fireable transitions. It also proposes a variable implementation model for a better adjustment to the given implementation environment. Several approaches were dedicated to the exploitation of the protocol inherent parallelism [Stei93],[Fisc94],[Plat96]. The results obtained show that the efficiency of the derived implementation can be improved by using parallelism. However, the potential of parallelism in protocols is often too small and the additional overhead for synchronisation and communication too high to achieve a considerable efficiency gain. More promising results are expected from an integrating handling of layers and data operations. [Leue96] describes an algorithm for the derivation the *Common Path* from SDL specifications as a base for deriving implementations using the ILP approach. A compiler for deriving ILP implementations from Esterel is reported in [Brau96]. For standardized FDTs such compilers are still under development. However, recent research [Ahlg96] has pointed out that the application of ILP is not recommended in every case.

As long as there are no practicable solutions, practice goes other ways. For the SDL Cmicro-Code-Generator [Tele96], for example, the use of certain SDL constructs (object-oriented features, enabling conditions, continuous signals, *import/export, view/reveal* etc.) is entirely prohibited and the usage of other constructs (*save, create, output to parent/sender* ..) is recommend to avoid. Thus, the implementation overhead caused by the semantics of these language elements is reduced.

Considering these approaches, we see two principle ways to improve the performance of automatically generated implementations:

(1) to support the implementation process (restricted use of certain language features, new language concepts that provide better implementations, appropriate specification styles), and
(2) by improving the implementation techniques (mapping strategy, hardware/software codesign, variable implementation models).

In this paper, we consider aspects of both ways for the standardized FDT Estelle [ISO89]. In particular, we discuss how semantic constraints can influence the efficiency of the generated code. In the first part of the paper we discuss how certain language features may have a restraining effect on the performance of the implementation. The second part of the paper investigates the application of more efficient mapping strategies in the implementation models of the transformation tools. It discusses the use of different process models (server model, activity thread model). We show how the more efficient activity thread model can be used and discuss the constraints for its application to Estelle. The efficiency gain received by applying this technique is shown for implementations of the protocols XDT and XTP.

2 INFLUENCE OF THE SEMANTIC MODEL OF ESTELLE

The derivation of implementations from a formal description requires measures to guarantee the conformance between specification and implementation, i.e. to assure that the verified design is correctly implemented. The straight-forward approach to do this is by exactly implementing the semantics of the respective FDT [Held95], [Gotz96]. This reduces the probability of implementation errors and decreases validation efforts. On the other hand, the semantics of the FDTs require an additional implementation effort that result in an overhead compared to hand-coded implementations. In some cases this overhead could be avoided, because it is not needed for the description of the problem. For example, the dynamic generation of a module in Estelle requires the scheduling of the module according to the parent/children priority principle. To reduce this overhead it is necessary to be familiar with the impact of certain language features on a later implementation. In this section, we give two examples.

2.1 Module attributes

The module attributes play a significant role in an Estelle specification. They define the position and the behaviour of the modules within the module hierarchy, thus determining their relations to other modules as well as the manner how their transitions are selected for firing. The selection depends on the requirements of the application to be specified, but sometimes different choices are possible, i.e. it is the subjective decision of the specifier. The attributes selected may have an influence on the ef-

ficiency of the derived implementation. The reason for this is that according to the operational semantics of Estelle the modules are differently handled when selecting the next transitions for firing.To demonstrate this effect we have implemented the XTP protocol [Prot92] based on the Estelle specification of [Budk93] with different attributes. For this experiment, the Estelle specification was embedded into a test environment. This test environment contains at the second level of the module hierarchy the modules: *Application*, *XTP* and *Virtual Medium* (in the following we call these modules main modules). We have considered the following 4 variants:

- Variant A (*systemprocess*):specification *non-attributed*, main modules *systemprocess*, child modules *process*
- Variant B (*process*): specification *systemprocess*, main modules *process*, child modules *process*
- Variant C (*systemactivity*): specification *non-attributed*, main modules *systemactivity*, child modules *activity*
- Variant D (*activity*): specification *systemactivity*, main modules *activity*, child modules *activity*

For the experiments, we used the connection-oriented unconfirmed data transfer service of XTP. We did not observe any changes of the service for these four attributations, i.e. no violation of the semantics of the specification was detected for this service. Due to the high complexity of the specification the influence of these changes of the attributes for the whole specification was not studied in detail for this experiment. The execution times measured for these alternative attribute variants are represented in Figure 1.

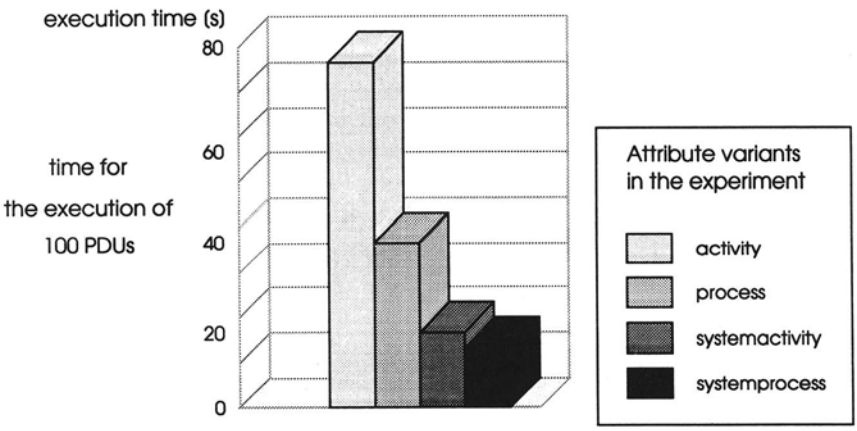


Figure 1 Impact of Estelle attributes on the execution time of an implementation.

The specifications were implemented by means of the PET/DINGO tool [Sije93] on a SUN SPARC 10 with the SunOS 4.1.3. operating system. The modules were mapped as follows: all modules up to the second hierarchy level on operating system processes, all modules below on procedures of the respective parent modules. The results of the measurements point out that the performance loss caused by the semantic constraints (compare the attribute variants *process* with *systemprocess* and

activity with *systemactivity*). The nondeterministic execution of transitions in modules attributed with *activity* causes a performance loss compared to the *process* variant. Note that the objective of these experiments was to demonstrate the influence the selection of an attribute may have on the implementation. The results depend of course on the implementation tool used and cannot be generalized. However, they point out that specifiers and implementors should prove their decisions on the influence they have on the further development steps.

2.2 Description and execution of transitions

The selection of the transitions for firing is defined by the operational Estelle semantics [ISO89]. It implements the parent/children priority principle and determines the execution of the specification. The used algorithm is very complex. It consists of two passes, one for determining the firable transitions and one for determining the executable ones. As shown in [Held95], up to 90% of the runtime can be spent on this selection. This overhead can be decreased by about 50% by an optimized algorithm that avoids the buffering of the fireable transition and thus needs only one pass.
The approach for selecting transitions in Estelle results in further consequences for the description and execution of transitions, which influence the execution time:

(1) The successor state must be known before executing a transition.
(2) All transitions of a module are considered for determining the firable transitions.

In SDL, for comparison, this is not required. The successor state is determined during the execution of the transition and all processes run concurrently. We illustrate the consequences of these differences for the execution time by means of the following example:

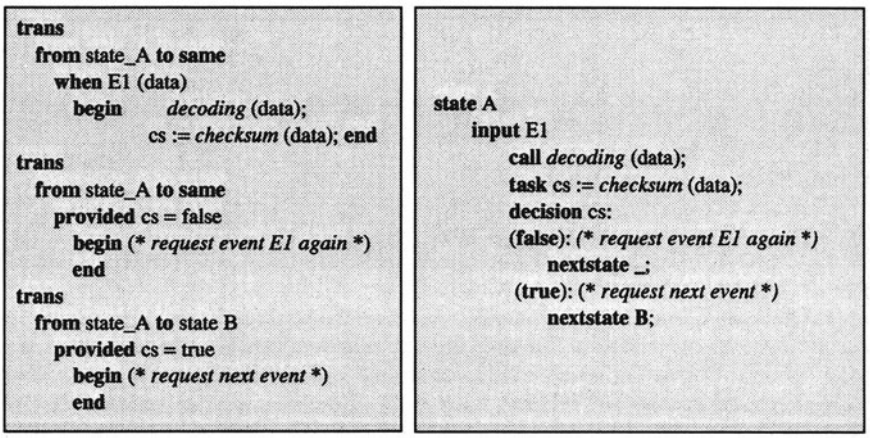
An event E1 is supposed to be executed in state A. For this, the data belonging to this event E1 have to be decoded and the checksum has to be calculated. If the checksum is correct, the entity changes in another state, otherwise it remains in the same state and demands the repetition of the transmission. The related specifications in Estelle and SDL are depicted in Figure 2.

The execution of event E1 (t_{trans}) requires the firing of 2 transitions in Estelle, whereas in SDL only one transition is needed. In addition, the transitions in Estelle have to be selected for execution according to the above mentioned algorithm (t_{sel}). In SDL the first firable transition is executed (on average $t_{sel}/2$). Thus, we get under the assumption that the time for the execution of a transition t_{trans} is equal in Estelle and SDL the following relation for the selection and the execution of a transition:

$$t_{estelle} = 2t_{sel} + 2t_{trans} \quad > \quad t_{sdl} = t_{sel}/2 + t_{trans}$$

The Estelle specification needs in this example $t_{trans} + 1{,}5t_{sel}$ more time than the SDL specification. The difference is even larger when the transition belongs to a child module and the parent module executes arbitrary often (n times) transitions before the child module can fire the second one (parent/children priority principle). This situation is even more extreme for a child module of an *activity*-module, because this module has to be nondeterministically selected between all child modules of the parent module. For this case, we get the following relation:

$$t_{estelle} = 2t_{sel} + 2t_{trans} + n(t_{sel} + t_{trans}) \quad > \quad t_{sdl} = t_{sel}/2 + t_{trans}$$



```
trans
  from state_A to same
    when E1 (data)
      begin      decoding (data);
                 cs := checksum (data); end
trans
  from state_A to same
    provided cs = false
      begin (* request event E1 again *)
      end
trans
  from state_A to state B
    provided cs = true
      begin (* request next event *)
      end
```

```
state A
    input E1
        call decoding (data);
        task cs := checksum (data);
        decision cs:
        (false): (* request event E1 again *)
            nextstate _;
        (true): (* request next event *)
            nextstate B;
```

Figure 2 Comparison of an Estelle and an SDL specification.

2.3 Summary of the discussion

The derivation of an implementation from a formal specification is determined by the formal semantics of the given FDT. The discussion above has shown that features of the semantic model of Estelle (e.g. transition selection rule, parent/children priority principle, static definition of the successor state) may have an adverse effect on the performance of an implementation. The specifier may in part benefit from this knowledge by applying these features with care. On the other hand, a general renunciation of the dynamic creation of modules would strongly restrict the descriptive power of Estelle. The application of asynchronous modules as proposed in [Bred94] could reduce the efficiency loss.

Note that our discussion does not touch the requirement of the independence of the specification from the implementation. It will only show that certain language features due to their semantics can adversely influence an automated implementation, if this is intended. This knowledge may help the specifier in choosing alternative representations. It can also be used for replacing parts of the specification by semantically equal representations when preparing and refining the specification for implementation.

3 INFLUENCE OF THE IMPLEMENTATION MODEL

Formal description techniques possess formal semantics that guarantee the exact interpretation of the specification. Automatically deriving implementations from formal descriptions requires transformation rules, which preserve the semantics of the specification, i.e. the conformance of the implementation with the specification. This adds overhead to the implementation, which is not present with implementations manually derived from informal descriptions. Besides these transformation rules have to be defined during tool development, i.e. they cannot take into account the respective implementation context. The set of these transformation rules forms the implementation model of the tool [Held95]. It defines the structure of the implemen-

tation, the interfaces, the process model and the relation to the implementation environment. The decisive factor is the process model, which describes the manner how the specification is mapped on the process structure of the operating system. In manual coding two principle approaches are applied: the server model and the activity thread model [Svob89].

Applying the server model, each system module (*systemactivity* or *systemprocess*) of the Estelle specification is implemented as a server that processes events (e.g. interactions or spontaneous transitions). Due to the semantic similarities between EFSM-based FDTs and the server model, current FDT compilers resort to the server model, because it allows a straight-forward mapping of the FDT semantics. All Estelle compilers known to us apply the server model.

The activity thread model implements the entities as a set of procedures. Each procedure implements a transition of the FSM. An incoming event activates the respective procedure, which immediately handles the event and when producing an output calls the respective procedure of the next entity. The sequence of the inputs and outputs (input→output→input ... input→output ...) results in a sequence of procedure calls - the *activity thread.* (Note that the term *activity thread* does not refer to a thread of the operating system. It denotes the execution path an event takes through the protocol stack.)

The server model exhibits extra overhead, which is not present with the activity thread model, e.g. overhead for asynchronous communication including queuing operations and overhead for process management. The problem with the activity thread model is that it is based on a semantic model that differs considerably from the semantic models of EFSM-based FDTs. We have shown in [Henk97] that the activity thread model can be applied to derive more efficient implementations form SDL specifications. In the following we want to discuss whether this model can be applied for automatically deriving implementations from Estelle specifications. We first give an overview how the transformation can be done in principle. Thereafter we discuss the constraints of this mapping.

3.1 Derivation of Activity Thread Implementations

In order to apply the activity thread model rather than the server model to derive code from protocol specifications, the following differences between the two models have to be taken into account:

- The active elements in the server model are the protocol entities. In general, each entity is implemented by a thread (or process) of the operation system or the runtime environment. The execution sequence of the events is determined by the scheduling strategy of the operating system or the runtime environment. Communication is asynchronous via buffers.
- The activity thread model is event-driven. The execution sequence of the events is determined by the sequence of inputs and outputs as given in the specification. Thus, the activity thread approach supports an optimal execution sequence of the events. Communication is synchronous, i.e. no buffers are employed. Because of that, the model is well suited for the layer-integrating implementation of protocols.

Estelle specifications do not allow a direct mapping on activity threads. In the following we present the principle of a mapping strategy, which comes close to the activity thread principle:

(1) Each Estelle module is transformed into a reentrant procedure that contains the executable code. With each procedure call, a single Estelle transition is executed. For each instantiation of an Estelle module, an Instance Control Block (ICB) is created. The ICB contains the following information:
   - all internal information (e.g. state information, local variables),
   - the addresses of the interaction points,
   - the address of the reentrant procedure.

   Each event is implemented by a structure containing the following data:
   - the event type,
   - the reference to the parameters of the event,
   - the next procedure to be called (address of the respective ICB).

(2) Each *output*-statement of an Estelle module is replaced by the call of the respective procedure, which implements the Estelle module that receives the event.

(3) For the whole specification, the following frame procedure is generated:

```
procedure AT (incoming_event)
        next_event := incoming_event;
        while not_empty(next_event) call_next_procedure (next_event);
```

The procedure *AT* is called by an incoming event. The procedure *AT* calls the procedures implementing the respective Estelle modules according to the structure of their *output*-statements. Output events sent to another Estelle module are stored in the variable next_event. The loop of the procedure *AT* is terminated when next_event is empty, i.e. the transition does not contain an output event. The termination of the loop also completes the activity thread, and the next incoming event can be handled. This frame procedure is necessary to map the state-oriented presentation of Estelle specifications onto the event-driven representation characterizing activity thread implementations. (Otherwise the specification has to be rewritten in an event-oriented style, which requires a new validation of the specification.)

The mapping strategy outlined above is depicted in Figure 3. It shows how the Estelle specification is mapped on the main procedure *AT*, which controls the execution of the activity thread, and the output event on procedure calls.

3.2 Problems and Constraints with Activity Threads

Applying the mapping strategy described above, some problems with the Estelle semantics arise. In the following we provide solutions how to preserve the semantics of the specification and discuss constraints existing for the application of the technique, respectively. We first discuss two problems, which are of general nature to this approach. After that we consider the problems, which specifically concern the Estelle semantics.

The general problems are:
- transitions performing an action after an output, and
- transitions with multiple outputs.

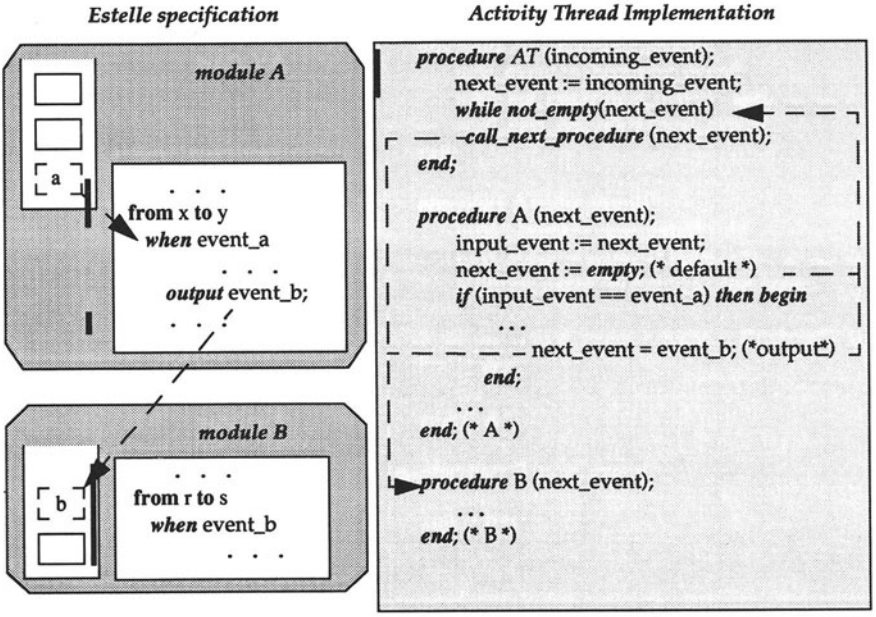


Figure 3 Basic strategy for the derivation of activity threads from Estelle specifications.

- **Transitions performing an action after an output**

The simple replacement of an output statement by the call of the respective procedure (implementing the Estelle module receiving the event) may violate the atomicity of transitions. An example for a possible violation of the atomicity of a transition is given in Figure 4. The figure shows two communicating modules, L2 and L3. For example, let us assume that L3 implements a routing function. An event e1, received by module L2 is routed to module L3. From L3 it is retransmitted to L2 and output as e2 to the medium. Within L2, the variable *dir* is employed to store the direction, in which the last event has been transmitted. The example shows that statements after the *output*-statement may produce a different behaviour as specified.

Our approach is to defer the procedure call until the transition has been completed. To ensure this the output event are stored in the variable next_signal of the frame procedure *AT* in section 3.2. The procedure *call_next_procedure* calls the respective procedure after the transition has been completed. From the viewpoint of the communicating modules this represents a transformation of synchronous communication to asynchronous communication. This is because the procedure implementing the receiving module is no longer called by the sending module itself. Instead, the procedure is called by the procedure *AT*.

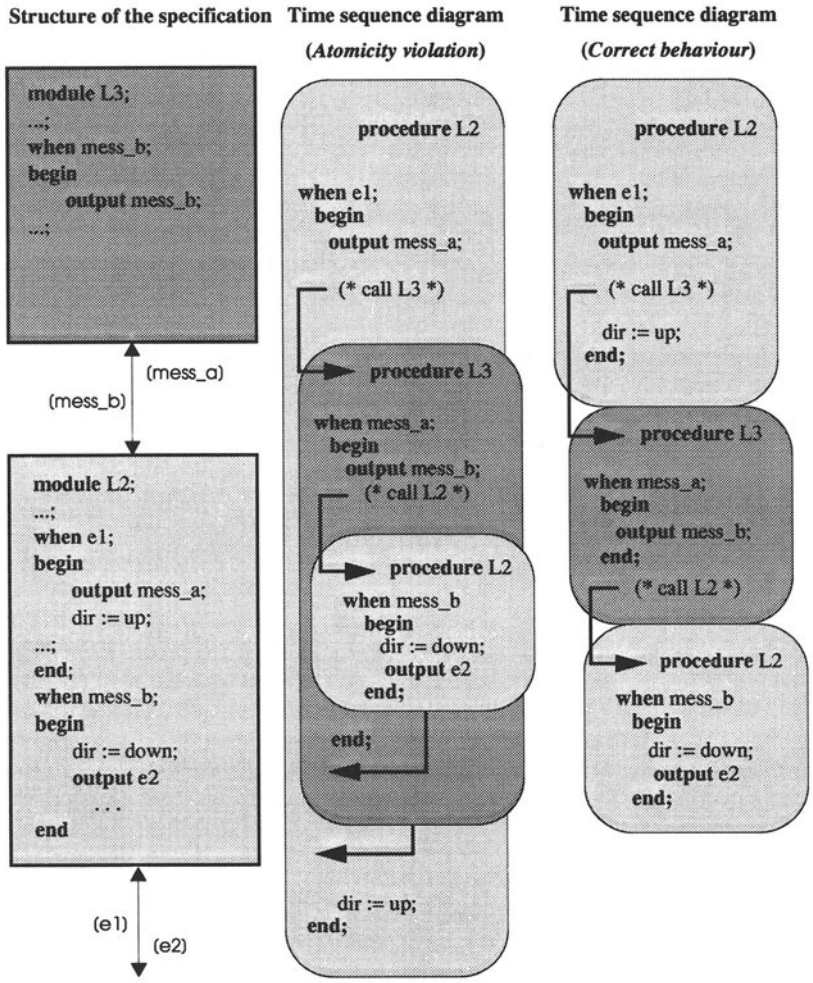


Figure 4 Example for the possible violation of the atomicity of a transition.

- **Transitions with multiple outputs**

A transition may contain several *output*-statements. In this case, a sequential call of the procedures, which implement the Estelle modules the events are sent to, is not always possible. Figure 5 shows an example for this. In the figure, the sequential execution of the two outputs of module P2 results in an overtaking of mess_c by mess_e. In other words, mess_e, which results from mess_d, reaches module P4 before mess_c. This clearly violates the semantics of the Estelle specification.

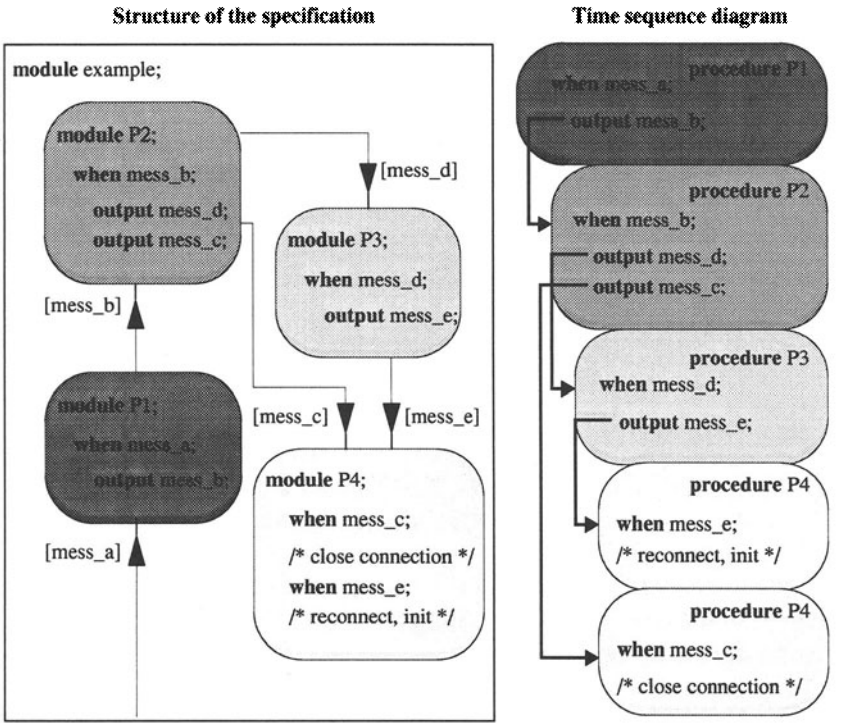


Figure 5 Example for the overtaking of events.

An approach to handle this problem is to split the activity thread in several activity threads. Thus, an additional activity thread is created for each additional output. To handle these additional activity threads, we have devised two process models, namely the *basic activity thread* model (BAT model) and the *extended activity thread* model (EAT model). The BAT model handles the additional activity threads quasi-parallely, whereas the EAT model processes them truly concurrently. As shown in [Henk97] BAT model implementations have in general proved to be more efficient, because they avoid additional communication and organizational overhead. Therefore, we focus on the BAT model in this paper.

For BAT model implementations the frame procedure *AT* is modified as follows:

```
procedure BAT (incoming_event)
        put incoming_event in AT_list
        while (list_not_empty(AT_list)) do
                for_all_elements AT in AT_list do begin
                        next_event := event_of(AT);
                        next_event := call_next_procedure (next_event);
                        if empty(next_event) then remove AT from AT_list;
                end
```

BAT model implementations handle only one incoming event per time, i.e. incoming events are always processed sequentially. No interleaving of the processing of incoming events is possible. Subsequent incoming events are blocked until the previous events have been processed.

Multiple outputs in a transition are handled by the creation of additional activity threads. One additional activity thread is created for each *output*-statement within a transition. These additional activity threads are managed by the AT_list. The management of the activity threads in the AT_list guarantees an execution order that is in accordance with the semantics of the specification. The AT_list is used to schedule the additional activity threads. Note that after the arrival of an incoming event, the AT_list contains exactly one activity thread. Several activity threads only emerge when a transition with multiple outputs is executed. In this case, the activity threads in the AT_list are scheduled according to the Round Robin principle. Only one procedure, i.e. one transition, is executed per iteration of the loop. Thus, the activity threads are executed in a quasi-parallel manner. When an activity thread terminates, it is removed from the AT_list. The main procedure *AT* returns after the last activity thread has terminated.

Now we discuss the semantic problems that are more specifically related to the Estelle semantics.

- **Parent/children priority principle**

With the activity thread model, the sequence of execution of the Estelle modules (schedule) is determined by the output executed by the respective transitions. Conversely, the Estelle semantics employs a complex algorithm to schedule the next transitions to be executed. This contradiction between the activity thread model and the semantics of Estelle can be only solved if the Estelle specification follows one of the following rules:

- within a subsystem, i.e. a system module with all of its descendants, at most one transition may be executable at a time or
- the child modules of *systemprocess* or *process* modules may not offer an executable transition at the same time any of its predecessor modules offers an executable transition.

This ensures that an executable transition of a child module can be actually executed instead of its parent, which would not be the case if the parent/children priority principle were applied.

In order to follow these rules, different approaches exist:

- If a system module has descendants, only the leaves of the hierarchy may be active. In case of descendants of type *systemactivity* or *activity*, only one child may exist (see Figure 6a).*
- The Estelle specification contains only system modules or equivalently, there are only asynchronous modules as proposed in (Bred94) (see Figure 6b).

* The activity thread model does not allow a nondeterministical selection of the next procedure to be called.

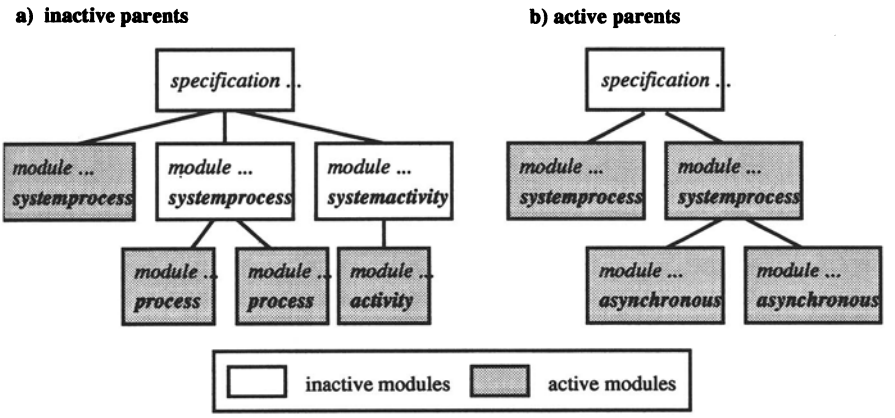


Figure 6 Examples of Estelle specifications.

In case the transition has a simple structure, i.e. input→action(s)→output, the transformation strategy described above can be applied. Thus, each output can be directly translated to a procedure call. transitions without output.

- **Spontaneous transitions**

Estelle allows spontaneous transitions, i.e. transitions that are not triggered by an event in a *when*-clause. Transitions with no incoming event cannot be implemented by an activity thread, since there is no event to be mapped on a procedure call. Spontaneous transitions have, therefore, to be removed during the refinement of the specification when preparing for implementation, or an alternative specification strategy has to be applied. If spontaneous transitions are used to model internal events, such an replacement can be usually easily done. The replacement of delayed transitions, however, is more complicated. In this case another specification strategy has probably to be considered. A possible solution could be to introduce timer modules, which are implemented using the *delay*-clause, as they are needed for real-life protocol implementations. Start and reset of the timer as well as a timeout can then be handled as incoming events and mapped on procedure calls.

4 QUANTITATIVE COMPARISON OF THE APPROACHES

In order to evaluate the performance improvements achievable with activity threads, we have manually applied our approach to derive code from XTP (Prot92) and the XDT protocol (eXample Data Transfer) [Koen96]XDT is an example protocol, which is used for teaching purposes in protocol engineering. It is a connection-oriented data transfer protocol supporting implicit connection establishment and data transfer based on the *go-back-N* principle. The XDT specification comprises about 350 lines. The Estelle specification of XDT follows the rules given in section 3.2. Thus, it supports the derivation of code according to the BATmodel.

In order to derive code from XTP, two specification variants have been used. The original XTP specification (rel. 3.6) we used comprises about 7900 lines (Budk93).

In order to apply the BAT model to derive code from the XTP specification, the Estelle specification has been adapted to follow the rules given in section 3.2. The adapted XTP specification is called aXTP in the following. Since the adaptation of the complete XTP specification to suit the BATmodel would have been an enormous effort, aXTP only supports the unconfirmed connection-oriented data transfer service. All other services have been removed from the Estelle specification. The resulting aXTP specification comprises 1000 lines. Deriving the aXTP specification from the original XTP specification, the spontaneous transitions have been replaced by introducing additional events in the input transitions. Note that the semantics of the unconfirmed connection-oriented data transfer service has not been changed by the modifications.

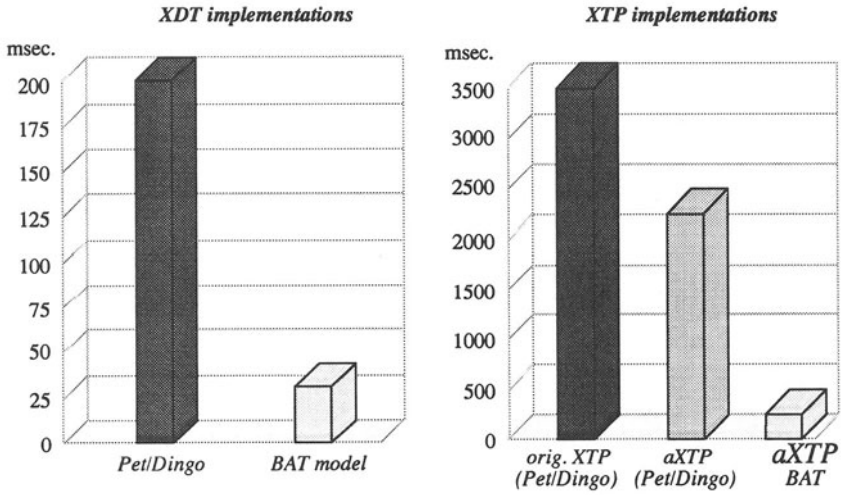


Figure 7 Quantitative comparison of the server model and the activity thread model.

For our experiments, we have derived code according to the BAT and the server model. For the derivation of the implementations according to the server model, we have used the PET/DINGO tools (Sije93). All the measurements were taken for a single connection of the respective protocol. The implementation platform has been a SUN SPARC 10 running the *Solaris* operating system (rel. 2.5). For the measurements, a test environment has been implemented that comprises the service user, which generates the data stream and measures its duration, and a virtual medium (i.e. an Estelle module) for realizing the communication between the protocol entities. The size of the PDUs used in the experiments was 512 bytes for XDT and 64 bytes for XTP. The XTP service we used was the connection-oriented unconfirmed data transfer. The measurements do not take into account the initialization phase. We have measured the times to process 50 PDUs by both peer entities. The results of the measurements are given in Figure 7. For XTP, the measurements for the original version as well as the aXTP version are given.

The measurements show that the activity thread implementations achieve a considerable speedup compared to the respective server model implementations. Comp-

ared to the server model implementation, the BATmodel implementation of aXTP exhibits a larger speedup as the respective implementation of the XDT protocol (i.e. a factor of 13.8 for aXTP and a factor of 6.7 for XDT). The difference results from the larger number of Estelle modules employed by aXTP, i.e. 17 versus 5 modules of XDT. This results in a larger number of queuing operations needed for the server implementation of aXTP.

5 CONCLUSIONS

The performance of implementations automatically derived from formal specifications depends on many factors. The inefficiency of current implementations is in part caused by the overhead involved with the implementation of the FDT semantics. In the paper, we have described how these semantical constraints can influence the performance of implementations derived from Estelle specifications. This impact concerns both the application of certain language features and the use of efficient mapping strategies.

In Estelle, additional overhead is caused by the hierarchical approach to select an executable transition. We have presented measurements for XTP, which show that a speedup can be achieve if the attributing principle employed in the specification is chosen with care. In addition, we have analyzed the times to select an executable transition and have compared them with the respective times needed in SDL. The comparison shows that the semantics of Estelle add some extra overhead, which is not present in SDL.

In the main part of the paper, we have investigated the application of the activity thread implementation strategy elaborated in [Henk97] to Estelle. Our measurements for the two protocols XTP and XDT as well as the measurements made for SDL in [Henk97] show that with the activity thread approach a considerable speedup can be achieved. However, compared with SDL, which has only some minor semantic constraints to this approach, the application of the technique to Estelle requires severe limitations, which reduce the descriptive power of the language (as, for instance, a renunciation on the parent/children priority principle). Therefore, we currently focus on the implementation of a prototype compiler for SDL. On the other hand, automated implementation techniques from formal descriptions will be only applied in practice, if the efficiency of the generated code is close to the hand-coded one. Experiment reports like this one may help that FDTs move towards a better support of efficient implementation techniques.

6 REFERENCES

Ahlg96] Ahlgren B., Bjorkman,M.;Gunningberg P.: Integrated Layer Processing can be hazardous to your performance. In Dabbous,W.; Diot,C. (eds.): Protocols for High-Speed Networks V. Chapman&Hall, 1996, 167-181.

[Brau96] Braun, T. et al.: ALFred - An ALF/ILP Protocol Compiler for Distributed Application Automated Design. Rapp. de Recherche No. 2786, INRIA, 1996.

[Bred94] Bredereke J., Gotzhein R.: Increasing the concurrency in Estelle. In Tenney R. L., Amer P. D., Uyar M. Ü. (eds.): *Formal Description Techniques VI*, Elsevier Science Publishers,1994, pp. 26-29.

[Budk93] Budkowski S. (ed.): Formal Specification, Validation and Performance Evaluation of the Xpress Transfer Protocol (XTP). INT Evry (France), Research report No 931004, 1993.

[Clar89] Clark D.D. et al.: An Analysis of TCP-Processing Overhead. *IEEE Communications Magazine*, June 1989.

[Clar90] Clark D.D., Tennenhouse D.L.: Architectural considerations for a new generation of protocols. *ACM SIGCOMM*, 1990, pp. 200-208.

[Fisc94] Fischer S., Hofmann B.: An Estelle Compiler for Multiprocessor Platforms. In Tenney R. L., Amer P. D., Uyar M. Ü. (eds.): *Formal Description Techniques VI*, Elsevier Science Publishers,1994,pp.171-186.

[Gotz96] Gotzhein R. et al: Improving the Efficiency of Automated Protocol Implementation Using Estelle. Computer Communications 19 (1996), pp. 1226-1235.

[Held95] Held T.; Koenig H.: Increasing the Efficiency of Computer-aided Protocol Implementations. In Vuong S., Chanson S. (eds.): *Protocol Specification, Testing and Verification XIV*, Chapman & Hall, 1995, pp. 387-394.

[Henk97] Henke R., Koenig H., Mitschele-Thiel A.: Derivation of Efficient Implementations from SDL Specifications Employing Data Referencing, Integrated Packet Framing and Activity Threads. Accepted for *SDL Forum 97*, Evry, Sept. 1997 (to be published by Elsevier Publishers).

[ISO89] ISO IS 9074: Estelle - A Formal Description Technique Based on an Extended State Transition Model, 1989.

[Koen96] Koenig, H.: eXample Data Transfer. Technical Report 4/96, Department of Computer Science, BTU Cottbus, 1996.

[Leue96] Leue,S.,Oechslin,P.: On Parallelizing and Optimizing the Implementation of Communication Protocols.IEEE/ACM Trans. on Networking, 4(1), Febr. 1996.

[Plat96] Plato R., Held T., König H.: PARES - A Portable Estelle Translator. In Dembinski P., Sredniawa M. (eds.): *Protocol Specification, Testing and Verification XV*, Chapman & Hall, 1996, pp. 383-399.

[Prot92] Protocol Engines Inc.: Xpress Transfer Protocol Definition. Revision 3.6, January 1992.

[Sije93] 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.

[Stei93] Steigner C., Joostema R., Groove C.: PAR-SDL: Software design and implementation for transputer systems. Transputer Applications and Systems '93, R. Grebe et al. (Ed.), vol. 2, IOS Press.

[Svob89] Svobodova L.: Implementing OSI Systems. IEEE Journal on *Selected Areas in Communications*, 7(7),1987, pp. 1115-1130.

[Tele96] Telelogic Malmö AB. SDT Cmicro Package - Technical Description, Telelogic, 1996.

[Wats87] Watson R.W., Mamrak S.A.: Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices. ACM Transactions on Computer Systems. 5(2),1987, pp. 97-120.