34

Derivation of efficient implementations from formal descriptions - issues, methods and conformance

H. Kremer

University of Twente, Fac. Informatica, TIOS Group
P.O. Box 217, NL_7500 AE Enschede, The Netherlands
phone: + 31 53 893703, fax: + 31 53 333815, Email: kremer@cs.utwente.nl

Keyword Codes: D.2.10, D.2.5
Keywords: Software Engineering, Design / Testing and debugging

Abstract: The design and realization of systems is a complex task in which different design issues play a role in different design phases. This paper concentrates on the design issues that play a role during one of the last design phases: the derivation of an efficient implementation for a particular environment. This efficiency, also called performance, is measured in terms of resource usage (CPU and memory) and Quality of Service parameters (e.g. delay and throughput) for the data streams involved.

The design issues that arise during the implementation process are studied using a small system (a duplex 1-slot repeater) and a method for structuring systems that takes these design issues into account is presented. For early assessment of the effect of design decisions on the performance of the realization, Petri-Net based performance analysis techniques are used. A method is given for mapping LOTOS specifications via Extended Transition Systems onto Petri-Nets.

The results of this study include a method for deriving an efficient implementation in a sequential programming language from a LOTOS specification. The usability of the method is demonstrated, and a method is given for LOTOS based performance analysis and new conformance relations.

1. Introduction

The design and realization of systems is a complex task in which different design issues play a role in different design phases. In the first phases, the issues are related to capturing and satisfying the needs of the users, while in the last phases, the issues pertain to making an efficient realization for a particular realization environment. This efficiency (also called performance) is measured in terms of resource usage (CPU and/or memory) and Quality of Service parameter as delay and throughput.

This paper concentrates on the design issues that play a role in the design phase just before mapping a specification in a formal description technique (FDT) onto a realization in software. During this design phase, the issues that play a major role are i) taking characteristics of the procedure calls into account and ii) the removal of non-deterministic choice between interactions. A method for structuring systems that takes these issues into account is presented.

Understanding of the efficiency-effect of design decisions is crucial for making efficient realizations. An inefficient protocol has only inefficient realizations, but an efficient protocol is easily implemented and realized inefficiently. For example, an ill-designed timer package can contribute for more than 50% to the total time needed for processing a packet [3].

This method leaves the designer a lot of freedom in making design decisions. Performance analysis techniques allow the designer to asses the influence of his design decisions on the

D. Hogrefe et al. (eds.), *Formal Description Techniques VII*

efficiency of implementations (and realizations). A method is presented for mapping a LOTOS specification onto Petri Nets. These nets are analyzed using the SPNP [2] package whereby estimates for delay and throughput are obtained.

The contribution of this paper is the integration of FDT based implementation with the consideration of efficiency and with performance analysis techniques using Petri Nets.

A running example in the form of a duplex 1-slot repeater is be used throughout the paper for illustration of the design issues and demonstration of the usability of the presented methods.

The following restrictions are put on the specification that is used as a starting point:

- it consists of one or more independent (interleaved) components. This may seems a major restriction, but this isn't the case. Most multiprocessing operating systems do not allow processes to interact directly with each other: all interactions are via the operating system kernel. This kernel is part of the environment of the realization. When independent components of a system are mapped onto different processes, there is no need for direct interaction between these components as that interaction cannot be implemented.
- no dynamic process creation is allowed. No problems are anticipated with an extension of this work that allows dynamic process creation.
- the mapping of observable interactions onto procedure calls is present,
- it may not contain non-deterministic value generation.

This paper uses LOTOS as Formal Description Technique (FDT), but the results can be adapted easily for other process algebraic FDTs. The reader is assumed to be familiar with LOTOS, Petri-Nets [21] and the UNIX operating system.

1.1. Place in design trajectory

The design of a system, being the activity in which user requirements are formulated and translated into a real system, can be structured in a sequence of design phases. The Lotosphere design methodology [18] is taken as the methodology to which the work presented here belongs.

The four phases of this methodology are summarized as follows. The first phase is the *Requirement Capturing* phase in which the user requirement on the system are defined. The second phase is the *Architectural* phase in which a high-level description (called Architecture) is obtained that describes what the system is to do in terms of the future users of the system. In the third phase, the *Implementation* phase, a design is obtained (called Implementation) that can be mapped directly onto a Realization (see below) while preserving the properties of the Architecture. The Implementation describes how the system meets the User Requirements. The fourth and last phase is the *Realization* phase in which the Implementation is mapped onto the Realization. The phases of this methodology are described in more detail in [16].

The Implementation Phase can be divided into two steps. In the first step, the Architecture is transformed into a high-level design that describes how the Architecture is provided and that is realization independent. In the second step, the realization dependent aspects are taken into account The work described in this paper pertains to the second step of the Implementation Phase.

1.2. Outline

The outline of this paper is as follows.

In section 2, the design issues that play are role during the last steps of the second part of the Implementation phase are identified and a method is presented for structuring systems that takes these issues into account.

This method leaves a large design freedom to the designer. The designer can be aided in making decisions by using performance analysis techniques for assessing the influence of his design decisions on the performance of the realization without actually building the realization. In section 3 it is shown how LOTOS descriptions can be mapped onto Petri Nets that are subsequently used for numerical performance analysis. The results of this analysis show the effects the implementation decisions have on the mean delay and throughput.

In section 4 the conformance of the implementations to the specification is addressed. It is shown that, although the implementations are not testing equivalent with the specification, they are to be considered conforming implementations. A new conformance relation is defined that captures precisely the freedom the designer has during the implementation phase.

Section 5 discusses future work and the relationship with other work.

Section 6 contains the conclusions.

1.3. Example system and realization environment

The system used as a running example is a duplex 1-slot repeater. This system consists of two independent simplex 1-slot repeaters, each serving a stream of packets. A simplex 1-slot repeater is an active entity that contains a 1-slot buffer in which packets are received and are subsequently transmitted from. The behaviour of the duplex repeater is best described by the LOTOS specification in Specification 1. Note that the independence of the repeaters is modelled by interleaving of the repeaters.

Apart from the behaviour description there are no other constraints on the realizations.

```
Duplex [r1, w1, r2, w2]
where
  process Duplex[r1, w1, r2, w2] : noexit:=
  Simplex [r1, w1] ||| Simplex [r2, w2]
  endproc (* Duplex *)

  process Simplex[r, w] : noexit:=
  r?p : Packet  ;  w!p  ;  Simplex [r, w]
  endproc (* Simplex *)
```

Specification 1 *Duplex 1-slot Repeater*

Example realizations are made in software in the C programming language [15] on a system running the UNIX operating system. The following mapping of interactions is common practice, so it is presented without motivation. The interactions r_1, w_1, r_2, w_2 are mapped onto the functions calls `read(1,...)`, `write(1,...)`, `read(2,...)` and `write(2,...)` respectively. It is assumed that the write call is 'non-blocking'. This term is explained in detail in section 2.

2. Implementation issues and structure

In this section, some implementation issues that arise during the implementation process are identified, and a method is presented for structuring the specification such that the decisions made regarding the issues are reflected.

2.1. Implementation issues

This subsection contains the identification of implementation issues that arise during the implementation process.

Implementation issue 1 Allocation of independent components onto processes

The specification that serves as starting point consist of a number of independent (interleaved) components. As these components are independent, they can be executed in parallel. The number of processes onto which the independent components are mapped may be smaller than the number of independent components. A decision on the allocation of components onto processes must then be made.

Implementation issue 2 Avoiding choice between blocking call-instances

The default semantics of the read and write calls in UNIX are that they are *blocking calls*: these calls returns only when all activities related to the call have been successfully performed. A read call is successful only when data is read, so this call returns only if some data is actually received.

The notion of 'all activities related to a call' is vague and whether they have completed is not visible to the initiator of the call. A more precise characterization of a blocking call is the following. A call is a *non-blocking call* if the call always returns within a reasonable small period of time; it is called a 'blocking call' otherwise. An instance of a blocking call can be either blocking or non-blocking. It is the latter if it will return within a reasonable small period of time.

In the duplex repeater realizations, it is assumed that the write calls do not block, so only blocking in read calls can occur. If a realization in a single process initiates a read call for one stream and a packet of another stream is ready to be received, then the realization is unable to receive this packet as it is 'blocked' in the read call. It is clear that in this situation, the performance of the realizations is less than optimal as receipt and transmission of a packet is unnecessarily delayed.

This poor performance is caused by the choice the realization is forced to make a between interactions without knowing whether a packet is present that can be received or not.

Implementation issue 3 Executing multiple interactions between scheduling

The execution of poll together with the choice of which interaction is to performed next is called scheduling. Scheduling takes a small amount of time that is generally small compared with the time needed for interactions, but this amount is not negligible. Therefore, it is advantageous to schedule interactions only when necessary: if a non-blocking call can be initiated as an alternative to a blocking call, initiating the non-blocking call is often advantageous.

In the duplex repeater, write calls are never blocking, so after completion of a read call, the related write call can always be performed without need for scheduling, thus improving the performance.

Implementation issue 4 Removal of non-deterministic choice

Specifications generally contain non-deterministic choice between interactions. In realization environments, all processes are fully deterministic. During the implementation process any non-deterministic choice needs to be removed. The way this is done affects the performance of the realizations.

During the execution of the realization, the following sequence of interactions can happen: initially, both slots are empty and a packet is received at one of the interaction points, say r_1.

Immediately after that, a packet is ready to be received at r_2. At this moment, the realization can either i) receive that packet at r_2 and transmit them in arbitrary order or ii) postpone the receipt of the packet at r_2 until after transmission of the packet previously received at r_1.

The differences between both alternatives are that in i) both the maximum amount of memory used and the delay of packets is smaller.

2.2. Structuring systems

The previous subsection has illustrated the effects implementation decisions have on the efficiency of the system. In this subsection, a method is presented for structuring systems that takes the efficiency issues into account.

Rather than taking all issues into account in a single step, the structuring of systems is divided into a sequence of steps, each considering some of the issues. The motivation for the ordering of steps is given in section 2.3. The steps are described in the chronological order in the subsections below.

2.2.1. Step 1 - Allocation of independent components to processes

If multiple processes are available, the designer has the freedom of allocating one or more of the independent components to a process. Allocating each component to a separate process has the disadvantage of the high cost of switching of the CPU between the processes.

In the remainder it is assumed that the duplex repeater is realized using a single process because this is more illustrative of the design issues.

2.2.2. Step 2 - Identification of blocking interactions

In order to be able to reason about blocking of interactions, an attribute is associated with each interaction. The boolean value of this attribute indicates whether the procedure call with which the interaction corresponds is blocking.

In the duplex repeater, the attribute is true for all r interactions and false for w interactions.

2.2.3. Step 3 - Structuring the specification into threads

Implementation issue 3 motivates the execution of behaviour with more than one interaction between instances of scheduling. This can be reflected in the specification by structuring the specification in special behaviours called threads. A thread is a behaviour that starts with a single interaction and that contains at most one blocking interaction. If a blocking interaction is present, only the starting interaction is blocking.

In the duplex repeater, the following threads can be identified: an r_1 followed by w_1 and a r_2 followed by w_2. Another set of threads consists of four threads, each consisting of a single interaction. The choice between the possible sets of threads is a matter that the designer must decide on. Note that in general, a large number of ways for structuring a specification into threads is possible. A structuring into threads of the repeater can be reflected in LOTOS as shown in Specification 2.

```
behaviour
state1 [poll, r1, w1, r2, w2]
where
  process state1[poll, r1, w1, r2, w2] : noexit:=
    r1?p : packet  ;  w1!p  ;  State1 [poll, r1, w1, r2, w2]
       []
    r2?p : packet  ;  w2!p  ;  State1 [poll, r1, w1, r2, w2] )
  endproc (* state1 *)
```

Specification 2 *Threaded specification*

In general, a large number of differently structured specifications can be obtained in this design step.

2.2.4. Step 4 - Addition of poll interactions

Most multiprocessing systems offer a facility that allows a process to determine whether an procedure call instance would block if initiated now. In UNIX, this facility is called poll or select. Using this call, a process can replace the choice between blocking call-instances by a single blocking call, followed by a choice between non-blocking call-instances.

According to Implementation issue 2, no choice should be made between blocking interactions in a single process when it is not certain that they do not block if initiated now. This choice, if present, can be avoided by introducing a special interaction (that is called poll and is mapped onto the poll procedure call) that allows a system to interact with its environment The value established in this interaction allows identification of the interactions whose procedure calls do not block at this moment.

2.2.5. Step 5 - Removal of non-determinism choice

The removal of non-determinism is done in this last step. General techniques for elimination of non-deterministic choice are making random choices and assigning priorities to some interactions over others. The representation of making a random choice in a LOTOS specification is shown in Specification 3.

```
behaviour
state1 [poll, r1, w1, r2, w2]
where
  process state1[poll, r1, w1, r2, w2] : noexit:=
  poll!Set(r1, r2) ?NotBlock : Interaction_denotations  ;
     (let e : interaction_denotation = Random(NotBlock) in
      [e = r1] -> r1?p : packet  ;  w1!p  ;  State1 [poll, r1, w1, r2, w2]
       []
      [e = r2] -> r2?p : packet  ;  w2!p  ;  State1 [poll, r1, w1, r2, w2] )
  endproc (* state1 *)
```

Specification 3 *Random choice between r_1 and r_2*

Notes
a) An interaction_denotation is one of a set of constants that identifies one or more interactions
b) The operation Set creates a set of interaction_denotations
c) The operation Random randomly selects an interaction from its parameter.

2.3. Discussion

This section so far has presented a method that was presented "as it is": it was not discussed why this particular method was chosen. It is easy to see that the steps in the method are related to the design issues. The motivation of the ordering of steps is given below.

An important issue is the avoidance of choice between blocking interactions. Thus the identification of which interactions are blocking is one of the first design steps. Allocation of parallel components onto processes was done as a first step, but could be done after step 2) as well, but Step 1 (Allocation of independent components onto processes) must precede step 3) because only within one process the choice between blocking interactions should be avoided.

Step 3 (Structuring the specification into threads) should always precede Step 4 (Addition of poll interactions) because in the reverse order a poll interaction might to be added in a choice between a blocking and a non-blocking interactions. If the non-blocking interaction is always preferred, then the poll interaction never occurs. Hence the addition of the poll interactions was unnecessary. The presence in a specification of an interaction that never happens is a violation of the propriety design criterion [16]. The addition also increases the size of code.

Step 5 (Removal of non-deterministic choice) can be incorporated partly in step 3) (structuring into threads) party in Step 4 (Addition of poll interactions). There are no definitive arguments for preferring either way so for personal preference, the removal of non-determinism is presented as a single design step.

Step 4 (Addition of poll interactions) should be done after Step 3 (Structuring the specification into threads) as, in order to avoid the choice between blocking interactions, knowledge of which interactions are blocking must be available.

3. Performance analysis

The method purposely leaves the system-dependent decisions that effect the performance to the designer: the allocation onto processes, the identification of threads and the removal of non-deterministic choice. This section shows how performance analysis techniques can be used for assessment of the efficiency of the realization without actually building it.

Being able to estimate the performance of a realization without building it is necessary when building realization is expensive (e.g. realizations in hardware), or when realizations are easy to build, but the realizations' environment difficult to influence.

For assessment of the performance of realizations the time needed for the execution of an interaction is an important measure. This time is called the duration of the interaction.

3.1. Approach

The approach taken here involves modelling the system and its environment as stochastic Petri-Nets. The performance of these nets is analyzed using the SPNP package [1]. Petri-Nets were chosen because they are well-known and capable of expressing concurrency and because of the availability of tools for performance analysis of Petri-Nets. SPNP was chosen because it was available locally.

A number of scenarios are defined that represent the outcome of different implementation decisions. These decisions vary in the number of processes and memory locations used and the scheduling of interactions.

The scenarios will be analyzed for different arrival rates of packets. Performance parameters of interest are mean delay and mean throughput of packets. The parameters are obtained for identical arrival rates for both streams that vary from 50 to 700 packets/s.

Performance parameters that could be of interest but that are not studied here are memory usage, processor usage and the variation of delay and throughput.

3.2. Modelling using Petri Nets

The modelling of the system and its environment can be separated into three parts: i) modelling the particular environment in which the system is to operate, ii) modelling the behaviour as described by the specification of the system and iii) modelling the allocation of resources (process and memory) and the scheduling of interactions. The motivation for this division into three parts is that it separates the modelling of the environment from the modelling of the system and that it divides the modelling of the system into a part that depends only on the specification and a part that depends only on the implementation decisions.

For the analysis, it is assumed that the environment acts as a source of packets that are put in a queue from which they are received by the system. The environment also acts as a sink for the packets sent by the system. No queue is necessary here. Both streams of the repeater are independent, so the environment is modelled as two sources of packets. The number of

tokens in the environment (10) is such that there are always tokens in the environment. The Petri Net of the environment is shown in Figure 4.

The specification depended part is derived from the specification of the system in two steps: mapping specification is mapped onto an Extended Transition System (ETS) and mapping the ETS onto a Petri-Net.

An ETS is basically a Labelled Transition System extended with variables. A full definition of ETSs and an algorithm for mapping LOTOS onto ETSs is found in [6,7]. The structure of the specification in terms of parallel components is to be preserved, so an ETS is made for each of the parallel components in the specification. The derivation of the ETSs for the duplex repeater is simple, so the ETSs are presented without argumentation in Figure 5.

Due to the nature of Petri Nets, any choice between interactions within the same ETS may not depend on the values established in previous interactions. If such choice is present, it needs to be replaced by a non-deterministic choice whereby the probabilities of the alternatives are known. It would be desirable to have tools for the analysis of Petri Nets that are extended with variables. An example of such nets is found in [9].

The mapping of ETSs onto Petri-Nets consists of two steps: mapping the interactions and mapping the ETS structure.

Transitions in Petri-Nets have no duration so an interaction with a non-zero duration must be mapped onto the Petri-Net |▸○▸|. If a token is present in the place then some processor is busy with processing of this interaction. The first transition is an immediate transition and models the start of the interaction. The second transition is a timed transition that models the end of the interaction and the rate of this transition models the duration of the interaction.

The structure of the ETS can be mapped by mapping each state of the ETS onto a set of places, one place for each of the variables present in state, and a set of arc connecting these places with the Petri-Nets onto which the interaction are mapped. The Petri-Net model of the duplex repeater is shown in Figure 6.

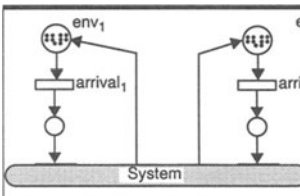

Figure 4 Model of the Environment

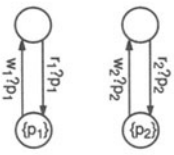

Figure 5 ETS of duplex repeater

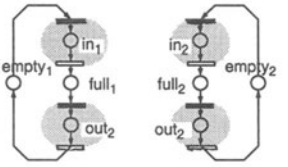

Figure 6 Basic model of the system

Note: The shaded areas in Figure 6 identify the subnets that model the interactions.

For each of the resources (processors and memory), a place is defined that indicates whether the resource is used or free.

Processors

For each processor (or process), a token is defined that, when present in the idle location represent the processor being idle. If a processor is busy executing an interaction, then the corresponding processor token is found at the place in that interaction. The start of a thread is modelled by an input-arc from the processor to the first interaction of the thread. Likewise, the end of a thread is modelled by an output arc from all last interactions of the thread to the processor.

If the processors can be freely allocated to threads, a single place represents all processors. If some processors are only allocated to some threads, then multiple places are needed for representing the processors.

Memory

For each memory location, a token is defined that, when present in the memory location represent the memory location being empty. Arcs present represent the allocation/deallocation of variables to memory. Any memory location to which a variable that is obtained in an interaction is mapped must be allocated before the interaction while any deallocation can only be done at the end of an interaction. The allocation/deallocation is assumed to occur in zero time.

3.3. Scenarios

A number of scenarios is defined in which the effect on the performance of the implementation decisions are analyzed. In Petri nets, only choices can be made between transition that are enabled, so the poll interaction need not to modelled as such.

scenario 1 is the baseline scenario. It involves threads consisting of single interactions. The choice between interactions is a random choice. The Petri Net model of the system and its environment according to this scenario is shown in Figure 7.

In **scenario 2,** the scheduling involves scheduling of threads rather than single interactions. The choice between the interactions is still a random choice.

Scenario 3 analyses the influence of non-negligible time needed for polling. An extra place and a transition are introduced between the place CPU and its outgoing arcs. The rate of the transition is determined by the time needed for scheduling.

Scenario 4 analyzes the effects of making a choice between blocking interactions. In this scenario, interactions are modelled by a different subnet than in the other scenarios. This subnet is shown in Figure 8.

For all of the scenarios non-deterministic choice is removed in two ways: either by random choice or by giving a fixed priority to interactions of stream 2. Note that giving a fixed priority to w interactions coincides in this system with the scheduling of threads consisting of two interactions.

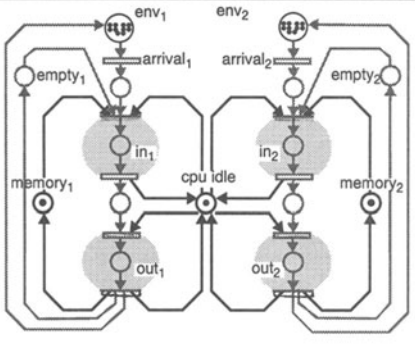

Figure 7 Full Petri Net scenario 1

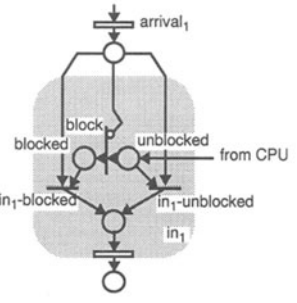

Figure 8 Interaction subnet scenario 4

Note: Figure 7 is obtained by superposition of figures 4, 6 and the allocation of resources

The rates of the timed transitions are based on the actual measurements presented in [3]. The processing time for r and w interactions for a packet with average length 1400 bytes is 926 μs. No estimate is available for the time needed for scheduling, so this time is arbitrarily set to 1/10 of the packet processing time.

For SPNP, only Negative Exponential distributions of parameters are allowed, so it is assumed that the all rates have a Negative Exponential distribution with mean equal to the given time average. In the remainder of this text, when used without and adjective, the terms delay and throughput denote the mean delay and mean throughput.

3.4. Results

All results are obtained using the SPNP version 3.1 on a Sun SPARC station IPX with 32MB memory, running SunOS 4.1.1 at 40 MHz. The total time for determining all results is approximately 13 minutes. This time includes no optimization other than those present within SPNP itself.

The measurements are shown in Figure 9 and Figure 10. The results of assigning priority to stream 2 in scenario 4 are not shown as the stream 1 is permanently starved by stream two.

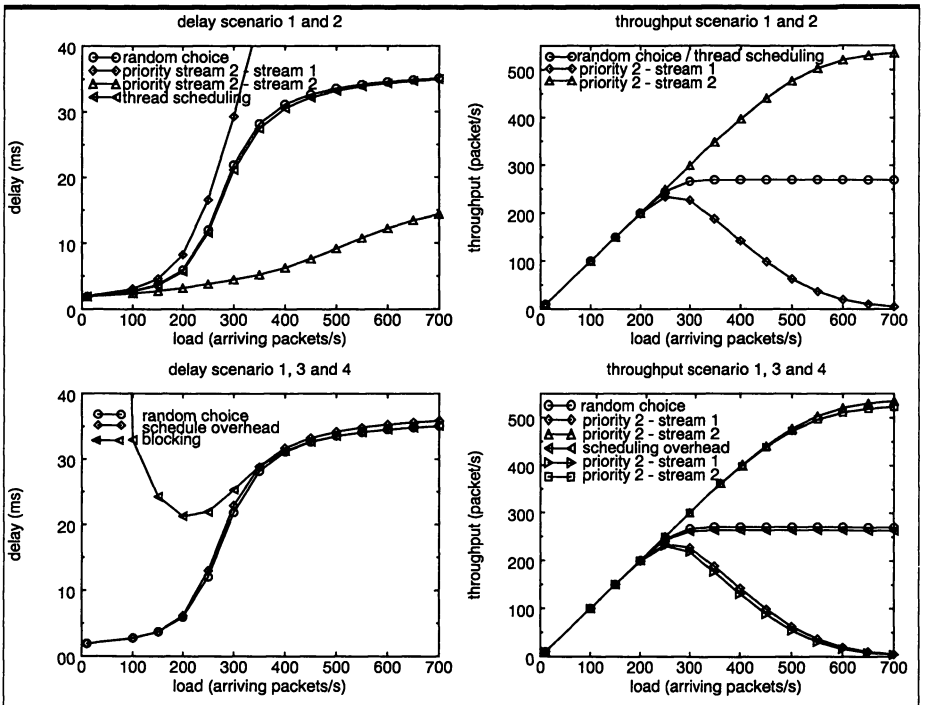

Figure 9 Performance results: mean delay

Figure 10 Performance results: mean throughput

Notes: a) Depicted are performance results for varying rates of arriving packets.
b) Unless indicated, measures are the same for both streams.

Based on these measurements the following conclusions can be drawn:

- Under low load, the difference in delay between scenarios 1 through 3 is very large. Under high load, the difference between scenario 4 and 1 is small
- Under low load (≤200 packets/s), the difference in throughput between all scenarios is small. Under high load (≥300 packets/s), assigning priority to stream 2 causes starvation of stream 1.
- The effects of a non-zero scheduling time are small under low load, but not negligible: at 200 packet/s the increase of delay due to scheduling is 3.5%.
- Making a choice between interactions without knowing whether they will block (scenario 4) severely affects the delay, so making avoidance of the choice between interactions a main design issue is justified.

3.5. Discussion

3.5.1. Results

The performance gain by scheduling threads with length greater than 1 rather than single interactions (Implementation issue 3) is a small (3.5%) while a higher gain might be expected, especially for larger systems. This unexpected result may be caused by the simplicity of the repeater system and the particular assumption on the scheduling time.

Current tools are capable of obtaining transition based results (e.g. throughput) and place based results (e.g. average number of tokens or probability of being empty). The mean delay must be calculated using Little Law. This is a drawback as only the mean delay and not its variation can be calculated in this way.

3.5.2. Assuming Negative Exponential distributions

The used analysis package requires that all rates have a Negative Exponential distribution. In practice, this distribution is very seldom observed. This subsection analyses the influence of assuming Negative Exponential distributions.

Negative Exponential distributions are assumed for the load offered by the environment and for the duration of interactions. For the load this is an acceptable assumption as any assumption is equally reasonable in the absence of more information on the nature of the distribution. For the duration, this is less acceptable as more information on the distribution is available: it is a discrete distribution when CPU speed and packet length are constant.

The Negative Exponential distributions were chosen as this choice facilitates numerical analysis of mean delay and throughput without need for simulation. If variation is also important then simulation, e.g. using UltraSAN [4], is necessary. Disadvantages of simulation are the less exact results and much larger analysis time [13].

The assumption relates to the aims of this paper in two ways. One of the aims of this paper was to show how performance analysis can be used during the implementation process. For this aim the precise distribution of load and duration are not relevant at all. Another aim of the analysis was to offer quantitative support for the design issues. Making the assumptions on the distribution of load and duration makes the quantitative results less accurate, but the relative ordering of results can be used to obtain insight in relative effects of the design decisions.

3.5.3. Obtaining the system model

The mapping of the LOTOS specification onto the Petri-Net was simple for the duplex repeater. The mapping consists of two steps: mapping the full LOTOS specification onto an ETS and mapping of the ETS onto a Petri Net. For the latter step, a mapping was given. The

first step (LOTOS → ETS) is much more complex. Fortunately, this activity can be performed by a tool (Smile [8])

A mapping of LOTOS onto extended finite state machines (XFSMs) can be found in [14]. In this mapping, a LOTOS behaviour is mapped onto a set of communicating XFSMs. The difference with our mapping onto ETSs is that the mapping onto XFSMs preserve the process-structure of the specification while the mapping onto ETSs flattens the structure. The motivation for performing the flattening is that is facilitates efficient mapping of the ETS onto the finite state model underlying programming languages.

The Petri Net models used are all without data. This disables direct modelling of deterministic choice if this choice depends on the contents of data that is associated with a transition, forcing the designer to make assumptions on the frequency of the choice between interactions.

4. Conformance

The conformance of the implementation to its specification must be ascertained. This is done by i) showing that the implementation of the duplex repeater are not testing equivalent with its specification, ii) motivating why these implementations are, although not testing equivalent with the specification, to be considered conforming implementations, iii) defining a conformance relation that captures precisely the freedom of the designer as identified in section 2 and iv) showing that the implementation conforms to the specification according to the newly defined conformance relation.

Showing that the specification and implementation are not testing equivalent is done by giving a counter-example. If the specification has performed an r_1 interaction, a choice can be made between the interactions r_2 and w_1. A implementation with priority to w interactions will never perform an r_2 interaction. Thus there is an observation of the specification that is never made of the implementation, hence specification and implementation are not testing equivalent. The implementation is a trace reduct of the specification.

There are two different conformance issues related to the conformance of the implementation to its specification: i) the implementation of two independent components in the specification within a single unit of parallelism and ii) the designer's freedom in giving priority to some interactions above others. The motivation for identifying them as separate conformance issues is that they are related to different design issues. The first conformance issue is related to Implementation issue 1 while the second conformance issue is related to Implementation issue 4. These implementation issues are independent from each other, so the corresponding conformance issues are also independent from each other.

The two conformance issues are formalized into two conformance notions that are derived from the well-known testing equivalence notion. Testing equivalence was chosen as starting point because the current design phase is close to realization.

4.1. Implementing independent components within a single process

This conformance issue is explained using scenario 1 with priority to w interactions. The duplex repeater system consists of two independent simplex repeaters that each receive a packet and transmit it with some delay. A realization of a simplex repeater is correct when it exhibits a sequence of alternating r and w interactions, irrespective of the length of the time interval between successive interactions. This implies that a combined realization of two simplex repeaters can delay the execution of a r_2 for any reason, including delaying it because a w_1 interaction is executed first. This execution of w_1 is permissible because w_1 stems from component that is independent of that of r_2.

This can be generalized as follows: for a specification consisting of a number of independent (interleaved) components, a implementation is conformant if it implements the behaviour of each of the components individually correct.

A formal conformance notion called Parallel testing equivalence ($\approx_{|||te}$) is defined below that captures this informal generalization.

Definition: $\approx_{|||te}$ Parallel testing equivalence

Given two (LOTOS) specifications `Imp` and `Spec` with

`Spec` = S_1 ||| S_2 ||| ... ||| S_n,

`Imp` $\approx_{|||te}$ `Spec` iff $\forall$ i = 1...n: $Imp_{|L(S_i)} \approx_{te} S_i$ whereby

L(S) denotes the set of observable interactions of S and

$Imp_{|X}$ denotes the restriction of the observations of `Imp` to only interactions contained in X. All interactions that are not element of X are simply removed from all observations of Imp. The restriction of the observation $r_1;r_2;w_2;w_1$ to the set $\{r_1, w_1\}$ gives the observation $r_1;w_1$.

This notion is weaker than testing equivalence which is easily inferred from the duplex repeater.

Note that $\approx_{|||te}$ also captures the conformance issue related to the addition of the poll interaction.

4.2. Choice between interactions

The second conformance issue is the way an implementation makes a choice between interactions.

Conformance testing theory (e.g. [23]) involves the notion of refusal of an interaction: an interaction is refused if in a sufficiently long period the interaction has not occurred. Likewise it can be said that a procedure call instance is refused when doesn't return within a sufficiently short period. i.e. the call instance is blocking. This leads to the conclusion that an interaction is refused when the corresponding procedure call instance is blocking.

During the implementation process it was assumed that some procedure calls would never block. This is equivalent with assuming that the corresponding interactions are non-blocking. This implies i) that any test of the realization involving refusal of a non-blocking interaction cannot be performed and ii) that the realization, when given the choice between a set of interactions, it can chose any interaction that is non-blocking.

An implementation that makes such choice in a fixed way is a trace-reduct of its specification (`Imp` $\leq_{tr}$ `Spec`). Rather than allowing any trace reduct of the specification to be a correct implementation, only implementations that a reduct with respect to the choice between non-blocking interactions are considered correct. For the duplex repeater realizations, this means that a realization whose observations consist of sequences of $r_1;w_2$ and $r_2;w_2$ is considered correct while a realization whose observations consist of sequences of only $r_1;w_1$ is not.

This notion of conformance is called Implementation Preorder modulo a set of interactions and is defined as follows.

Definition: $\leq_{imp,X}$ Implementation preorder modulo a set of interactions X

Given two LOTOS specification `Spec` and `Imp` then `Imp` $\leq_{imp,X}$ `Spec` iff
There exists a partitioning of `Traces(Spec)` into `Traces(Imp)` and a set of not observed traces `NOT(Spec,X)` whereby

The set NOT(Spec, X) is defined by the following condition:

$\forall$ v $\in$ Traces(Spec), $\forall$ a $\in$ L(Spec), $\forall$ w $\in$ L(Spec)*:
(v;a;w $\in$ NOT(Spec,X) and v;a $\notin$ Traces(Spec)) implies
$\exists$ x $\in$ X: v;x;w $\in$ Traces(Spec)

The NOT contains precisely the observations that can be made of the specification but that are not made of the implementation due to a choice between interactions made by the implementation.

Verification that $\leq_{imp,X}$ is indeed a preorder is done by showing that the conformance relation is reflexive (Spec $\leq_{imp,X}$ Spec) and transitive ($Spec_1 \leq_{imp,X} Spec_2$ and $Spec_2 \leq_{imp,X} Spec_3$ implies $Spec_1 \leq_{imp,X} Spec_3$). It is easily verified that this is indeed the case for fixed X.

As both conformance issues are independent, it is conjectured that the definition of $\leq_{imp,X}$ can be extended to the $\leq_{|||impl,X}$ conformance relation. This allows compositional verification of the conformance of an implementation to its specification. A second conjecture is that $\leq_{|||impl,X}$ captures precisely the designer's freedom.

It is easy to see that $\leq_{|||impl,X}$ is weaker than testing equivalence for non-empty X and that $\leq_{|||impl,X}$ and testing equivalence coincide for empty X.

4.3. Verification of conformance

The next thing to do is the verification of the conformance between the implementation of the duplex repeater to their specifications. This verification is done in a rather informal way and done separately for scenarios 1 through 3 and for scenario 4.

In scenarios 1 through 3, the implementations interact with the environment in a desired way: if the environment refuses either r_1 or r_2 or both, all implementations under all scheduling strategies perform some w_i interaction. As the environment is not able to refuse any other interactions, it suffices to conclude that all implementations conform to the specification modulo the set $\{w_1, w_2\}$.

In scenario 4, the situation depends on the scheduling strategy. If random choice is made, the scenario conforms in the same way as the other scenarios. Whether the performance of the scenarios is acceptable is an issue that is not captured by the conformance relation. Under the strategy with priority to interactions of stream 2, any interaction relating to stream 1 is never executed by the implementation, thus the observations of the implementation does not contain the observation r_1, w_2. As this observation is not part of NOT(Specification, $\{w_1,w_2\}$), this implementation is not conformant according to the $\leq_{imp,\{w_1,w_2\}}$ relation.

4.4. Discussion

This section has introduced new conformance notions that are weaker than testing equivalence. Why are these weaker notions used rather than using the well-known testing equivalence? The notion of testing equivalence fails to the design freedom into account that exists when a set of independent and non-deterministic specification components are implemented in a single unit of parallelism.

By making assumption on the environment in which the realization is to operate, the universe of environments is reduced. These assumptions are equally applicable to testers of the implementation because the testers are a particular environment of the implementation. This implies that a test involving refusal of a interaction whose procedure call is assumed to be non-blocking is a test that may not be performed. Performing such test does not discriminate between conforming and non-conforming implementations (which is what it should do),

but it discriminates the implementations that do respectively don't make the assumptions on the environment.

5. Relation with other work and future work

5.1. Relation with other work

Mapping of LOTOS onto Petri-Nets is not new. In [20] and [9] nets similar to Petri Nets are used. One of differences between their work and the work in this paper is that in this paper, the mapping is done via an intermediate level (Extended Transition System). The advantage is that by using ETSs, the mapping onto Petri Nets can be adapted more easily for process algebraic languages other than LOTOS. Mapping of other process algebraic languages onto Petri Nets are known from the literature [11,22,10] but the purpose of these mappings is to serve as a vehicle for either the definition of the semantics of the formal language and for simulation, rather than as a vehicle for performance analysis. A drawback of these languages (including LOTOS) is that the duration of interaction is not part of the semantics of the language so the mapping of the languages onto Petri Nets cannot take the duration into account. Recently an extension of process algebra's with durational interactions has been defined [12], but no mapping onto Petri Nets is yet defined for their language.

Other examples of the use of Petri Nets are [17] and [2]. The analysis described in the latter article is similar to the work in this paper, but they don't obtain the Petri Net from a formal specification but from an Ada description.

The work in this paper differs from the approaches taken by many authors when implementing using formal descriptions [24, 19]. Their approaches are more focussed on obtaining a rapid prototype of an arbitrary LOTOS specification than on obtaining an efficient realizations. Their design decisions are made in a pre-defined way, thus ignoring the effect of the implementations issues on the efficiency of the realizations.

5.2. Future work

The current mapping of LOTOS behaviour onto ETSs removes all structure present within the individual components. This sometimes is considered unwanted. Further work is needed that defines a mapping that preserve the process-structure present in the LOTOS behaviour onto some structure present in the C-code.

Insight in the validity of the restriction to Negative Exponential distributions is necessary. This can be gained by simulations or by measurements on realizations.

The need for two new conformance notions was identified and two formal conformance notions were defined. The applicability of these notions for other systems than the duplex repeater is not yet known, so this applicability needs to be studied.

6. Conclusions

This paper has presented a methodology for deriving efficient implementation out of a formal specification that has the mapping of interactions onto procedure calls defined and that contains no non-deterministic value generation.

Aspects of this methodology are i) the identification of implementation issues that affect efficiency, ii) a method for transforming a specification into an efficient implementation in which the designer is assisted by performance analysis, iii) conformance notions that captures the freedom the designer has during this transformation.

The methodology has been shown usable: the method for mapping a specification onto a implementation and the application of performance analysis techniques have been demonstrated using a small system.

Although the validity of the quantitative results in terms of average delay/throughput depend on the validity of the assumptions made on arrival and processing rates, these results can be used for comparing the effect of implementation decisions relative to each other.

Acknowledgments

Mark de Weger and Lex Heerink are thanked for commenting on drafts of this paper.

References

[1] G. Ciardo and J. K. Muppala. *Manual for the SPNP Package Version.* Department of Electrical Engineering, Duke University, Durham, NC, version 3.1 edition, Sept. 1991.

[2] G. Ciardo and J. K. Muppala. Analyzing concurrent and fault-tolerant software using stochastic reward nets. *Journal of Parallel and Distributed Computing*, 15:255–269, 1992.

[3] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen. An analysis of TCP processing overhead. *IEEE Communications Magazine*, pages 23–29, June 1989.

[4] J. Couvillion, R. Freire, R. Johnson, W. D. Obal II, A. Qureshi, M. Rai, and W. H. Sanders. Performability modelling with UltraSAN. *IEEE Software*, 8(5):69–80, Sept. 1991.

[5] M. Diaz and R. Groz, editors. *Formal Description Techniques — V.* North-Holland Publishing Company, 1993.

[6] H. Eertink. *Simulation Techniques for the Validation of LOTOS Specifications.* PhD thesis, University of Twente, Mar. 1994.

[7] H. Eertink, P. Kars, and H. Wennink. Reducing extended transition systems. Memoranda Informatica 93-09, University of Twente, Feb. 1993.

[8] H. Eertink and D. Wolz. Symbolic execution of LOTOS specifications. In Diaz and Groz [5], pages 295–310.

[9] H. Garavel and J. Sifakis. Compilation and verification of LOTOS specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification X*, pages 359–376. IFIP WG 6.1, North-Holland Publishing Company, 1990.

[10] U. Goltz. CCS and Petri nets. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes*, volume 469 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

[11] U. Goltz and A. Mycroft. On the relationship of CCS and Petri nets. In J. Paredaens, editor, *Automata, Languages and Programming*, volume 172 of *Lecture Notes in Computer Science*, pages 196–208. Springer-Verlag, 1984.

[12] R. Gorrieri and M. Roccetti. Toward performance evaluation in process algebras. Technical Report UBLCS-93-16, University of Bologna, July 1993.

[13] B. R. H. M. Haverkort. *Performability Modelling Tools, Evaluation Techniques, and Applications.* PhD thesis, University of Twente, Jan. 1990.

[14] G. Karjoth. Implementing LOTOS specifications by communicating state machines. In W. R. Cleaveland, editor, *Concur'92*, volume 630 of *Lecture Notes in Computer Science*, pages 386–400. Springer-Verlag, 1992.

[15] B. W. Kernighan and D. M. Ritchie. *The C programming language.* Prentice-Hall, 1988.

[16] H. Kremer, J. van de Lagemaat, A. Rennoch, and G. Scollo. Protocol design using LOTOS: A critical synthesis of a standardization experience. In Diaz and Groz [5], pages 231–246.

[17] C. Lindemand and F. Schön. Evaluation sequential consistency in a virtually shared memory system by deterministic and stochastic petri nets. In H. Schwetman, J. Walrand, K. Bagchi, and D. DeGroot, editors, *International Workshop on Modelling, Analysis and Simulation of Computer and Telecommunication systems*, volume 25 of *Simulation Series*, pages 63–68. ACM, 1993.

[18] Lotosphere. The lotosphere design methodology: Basic concepts. Deliverable Lo/WP1/T1.1/N0045/V04, Esprit Project 2034 "Lotosphere", 1992.

[19] J. A. Mañas, T. de Miguel, J. Salvachúa, and A. Azcorra. Tool support to implement LOTOS formal specifications. *Computer Networks and ISDN Systems*, to appear, 1992.

[20] S. Marchena and G. Leon. Transformation for LOTOS specs to galileo nets. In K. Turner, editor, *Formal Description Techniques*, pages 217–230. Elsevier Science Publishers B. V., Sept. 1989. Proceedings of the First International Conference on Formal Description Techniques.

[21] M. A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[22] E.-R. Olderog. Operational petri net semantics for CCSP. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of *Lecture Notes in Computer Science*, pages 196–223. Springer-Verlag, 1987.

[23] J. Tretmans. *A Formal Approach to Conformance Testing.* PhD thesis, University of Twente, Dec. 1992.

[24] A. Valenzano, R. Sisto, and L. Ciminiera. Rapid prototyping of protocols from LOTOS specifications. *Software Practice and Experience*, 23(1):31–54, Jan. 1993.