MASTER

System level performance modeling of a complex high-speed packet switch modeling PRIZMA-T using POOSL

Verhappen, M.

Award date:
1999

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
System Level Performance
Modeling of a Complex
High-speed Packet Switch

Modeling PRIZMA-T using POOSL

M. Verhappen

Carried out in a joint project with
the IBM Zürich Research Laboratory,
Communication Systems Division,
Rüschlikon, Switzerland.

Coaches: R.P. Luijten M.Sc. IBM Zürich Research Laboratory
         A.P.J. Engbersen Ph.D. IBM Zürich Research Laboratory
         J.P.M. Voeten Ph.D. Eindhoven University of Technology

Supervisor: Prof. M.P.J. Stevens

Date: November 30, 1999
Abstract

Current telecommunication systems should be able to provide services to a wide range of traffic classes. These classes have different characteristics, but should be handled in a uniform fashion, for example by one type of switch. While system complexity increases with time, demands on the time-to-market become more strict. There is no time to investigate all solutions. Therefore, an abstract system model is to be built that enables evaluation of certain system properties at an early development stage.

The first project objective is to obtain knowledge on how to build models of complex communication systems. The second objective is specification of a model of the PRIZMA-T switch. PRIZMA-T is a lossless, self-routing, single stage switch and is being developed at the IBM Zürich Research Laboratory. Some requirements for this system are lossless switching, the minimum availability of bandwidth and internal switch resources and an appropriate best-effort discarding scheme, all in the context of multiple traffic classes.

The model is specified in POOSL (Parallel Object-Oriented Specification Language), which is developed at the Eindhoven University of Technology. POOSL is a language with a fairly limited syntax but great expressive power. This leads to compact, discussible and efficient models. POOSL's mathematical semantics allow for formal qualitative and quantitative system verification.

The modeler should select a modeling approach from countless alternatives. To facilitate this choice, several general modeling issues should be related to the system under investigation. These issues are: a modeling view on real-time and functional behavior, communication and concurrency, decisions about the modeling of time and packet flows, parametrizability and collection and presentation of simulation results.

It is not possible to use analytical techniques for performance analysis of complex systems such as PRIZMA-T. The reason for this is the explosion of the system's state space. For empirical analysis, the Markov Chain Monte Carlo method is chosen and applied to the model. One simulation trace of the model provides a confidence interval for estimated metrics such as load, throughput, and delay. Future research should focus on confidence intervals for jitter and memory occupancy values.

After general modeling issues are considered, valid abstractions of the architecture specifications are made. Abstractions can be divided into intuitive abstractions and abstractions from architecture structure, communication and concurrency. This process is supported greatly by the expressive power of POOSL and its underlying system level design methodology. The abstract adequate system model, that results from the modeling phase described above, is able to evaluate system properties of PRIZMA-T and can be used to support design decisions for future PRIZMA generations.
## Contents

1 Introduction
   1.1 Project Context ........................................... 1
   1.2 Project Objectives ........................................ 2
   1.3 Thesis Outline ............................................ 2

2 PRIZMA
   2.1 General Introduction ....................................... 3
   2.2 PRIZMA-T Architecture ................................. 4
      2.2.1 Data Section ........................................... 4
      2.2.2 Control Section .................................... 5
   2.3 Modularity: PRIZMA-T Expansion Modes .................. 6
      2.3.1 Single-Stage Port Expansion ....................... 6
      2.3.2 Multistage Port Expansion ......................... 6
      2.3.3 Speed Expansion .................................... 6
      2.3.4 Link Paralleling ................................... 8
   2.4 PRIZMA-T Flow Control .................................. 8
      2.4.1 Problem Context ..................................... 8
      2.4.2 Problem Definition ................................ 10
      2.4.3 Previous Solution: Nested Thresholds .............. 11

3 POOSL
   3.1 System Level Design Methodology behind POOSL .......... 13
   3.2 Structural Entities of POOSL ............................ 14
      3.2.1 Process Objects ..................................... 14
      3.2.2 Clusters ............................................... 15
      3.2.3 Data Objects .......................................... 15
   3.3 Syntax of POOSL .......................................... 15
   3.4 Action Urgency: Two-Phase Execution Model .............. 16

4 General Switch Modeling Considerations .................... 17
   4.1 Models and Architectures ................................ 17
   4.2 Modeling View on Complex Real-Time and Functional Behavior .... 18
   4.3 Communication and Concurrency ......................... 18
   4.4 Modeling Time ............................................ 20
      4.4.1 Where to Model Time ................................ 20
      4.4.2 Packet Flow Modeling ............................... 22
   4.5 Parametrizability ....................................... 25
4.6 Acquisition and Presentation of Performance Data
4.6.1 Presentation of Results
4.6.2 Acquisition of Data
4.7 Conclusions

5 Performance Analysis
5.1 Definitions and Analysis Context
5.1.1 Definitions
5.1.2 Analytical versus Empirical Evaluation
5.2 Markov Chain Monte Carlo Analysis Method
5.3 Conclusions

6 An Abstract System Model
6.1 Introduction
6.2 Abstractions
6.2.1 Intuitive Abstractions
6.2.2 Architectural Structure Abstractions
6.2.3 Communication Abstractions
6.2.4 Concurrency Abstractions
6.2.5 Flow Control Related Abstractions
6.3 Modeling for Performance Analysis
6.4 The Model in POOSL
6.4.1 Sources
6.4.2 Input Adapters
6.4.3 PRIZMA-T
6.4.4 Output Adapters
6.4.5 Destination
6.4.6 Latency Process Class
6.5 Simulator Performance
6.6 Conclusions

7 Conclusions and Recommendations
7.1 Conclusions
7.2 Recommendations for Future Research

A POOSL Syntax

B POOSL Code Examples
B.1 Parallel Threads
B.2 Guards
B.3 Atomicity
B.4 Pluggable Mechanisms
B.5 Decision Moment Calculation
Chapter 1

Introduction

1.1 Project Context

Today's telecommunication systems offer a wide variety of services, such as the transport of voice, audio, video and data. These different services have very different traffic characteristics. Whereas telephony traffic is very regular, data traffic is generally very bursty. The telecommunications infrastructure however, should handle these traffic types in a unified fashion. Switches that route the traffic through the network, should therefore be capable of handling all these types of traffic, while maintaining a high degree of efficiency. In this way the network provider can comply with the throughput and latency constraints in the customer's traffic contracts using just one type of switch.

While customer bandwidth demand and switch complexity increase exponentially with time, demands on short product development times become more stringent. We have come to a point where development time is too short to unfold all conceptual solutions into the finest details and subsequently choose the best solution. The development of today's systems requires a different approach. In an early product design phase, we should investigate the impact of system level architecture decisions on system behavior. This investigation is supported by the construction of abstract system models.

An abstract system model should be used to answer certain questions about system properties. At the time of system specification, these questions and properties must be described in a precise fashion. With the properties in mind, the system level designer builds an adequate system model that has the right level of abstraction to answer just those questions. The answers provided by the model enable the modeler to decide on changes in the system's architecture.

The system level modeling approach in this thesis is based on the method Software/Hardware Engineering (SHE), developed by the Section of Information and Communication Systems of the Eindhoven University of Technology [1]. The system level modeling will be done in a simulator named SHESIM. In this simulator, model specification is performed in the language POOSL (Parallel Object Oriented Specification language).

In this project, we use the previously mentioned modeling approach to model a high-performance switch chip known as PRIZMA\(^1\). PRIZMA is a lossless, self-routing, single stage switch, developed at the IBM Zürich Research Laboratory. The next-

\(^1\) Packetized Routing in Zurich's Modular Architecture
generation switch chip, called PRIZMA-T, will be described in more detail in chapter 2.

The most important feature of the PRIZMA architecture is its performance modularity. The number of ports can be enlarged by connecting basic switching modules in a single stage or multistage network. Throughput can be increased by connecting several modules in parallel.

PRIZMA transfers fixed-size packets, of which the size is programmable. Its target is not just the switching of telecommunications traffic, such as ATM traffic and other packet based interconnections like IP and Ethernet traffic. PRIZMA may also operate as a switch in a multiple microprocessor environment. Since both traffic types and their concept of Quality of Service (QoS) are quite different, it is a challenge to support both in the same switch.

1.2 Project Objectives

This project has two main objectives:

- Building an abstract system model of the new PRIZMA-T switch. This project attempts to answer several questions about quantitative system properties of this switch and its environment. For that, we use the SHEsIM tool and the POOSL language. A more detailed description of these properties will follow in chapter 2.

- Building knowledge on system level modeling of complex communication systems. System level modeling is a relatively new research area that is becoming more important with exponentially increasing system complexity and decreasing development times. This thesis also aims to show a general approach to the modeling of such complex systems; mainly in the area of evaluation of performance properties. This may lead to useful extensions of the SHEsIM simulator, or the POOSL language.

1.3 Thesis Outline

Chapter 2 presents a description of the PRIZMA-T architecture and formulates questions about switch system properties within the preferred area of investigation. Next, chapter 3 briefly discusses the system level modeling language POOSL. General modeling considerations are discussed in chapter 4. Chapter 5 describes the properties to be analyzed and the theoretical and practical modeling approach to an empirical performance analysis method. An abstract system model and the actual modeling process are described in chapter 6. Chapter 7 presents conclusions on the modeling of PRIZMA-T as well as recommendations on the expansion of SHEsIM and POOSL. A few preliminary recommendations concerning PRIZMA-T are also discussed.
Chapter 2

PRIZMA

This chapter presents an overview of the PRIZMA switch system. First, it describes the switch and its environment. Subsequently, the architecture and expansion possibilities of PRIZMA-T are explained briefly. Finally, we discuss the need for flow control and the advantages and disadvantages of the flow control topology in the PRIZMA-T system. In the scope of traffic contracts that involve guaranteed traffic and best-effort traffic, we discuss the requirements that the flow control should satisfy. These requirements are: lossless switching, the minimum availability of bandwidth and internal switch resources and an appropriate best-effort discarding scheme, all in the context of multiple traffic classes. This chapter should be read as an introduction to the first part of our project objectives, which is the development of a model that is capable of supporting design decisions in the areas discussed above.

2.1 General Introduction

An overview of a single $N \times N$ PRIZMA switch [2] [3] [4] and its environment are depicted in figure 2.1 on page 4. Each input port connects to an input adapter that converts different types of packets into uniform fixed-length switch packets and inserts internal switch packet headers. Each input adapter also contains buffers for handling flow control information from PRIZMA. On the output side, an adapter is connected to every switch output port. The main function of these output adapters is buffering and traffic shaping. The buffers are needed in a speed escalated configuration in which the switch port speed is higher than the external line speed. The output adapter strips the internal switch header off the packets and performs a CRC check to evaluate proper switch operation. Each input adapter is connected to its peer output adapter for the exchange of control information (see section 2.4). Note that either of the adapters in this pair only has knowledge of its peer and is unaware of the status of other adapters in the switch system. The switch has to inspect the packet header to be able to perform the actual input to output translation. The statistical nature of the incoming flows may cause packets to contend for the same output port at the same time. Therefore, some packets must be buffered to resolve this contention. In PRIZMA, the packets are queued at the

---

$^1$ $N$ denotes the number of physical input ports and output ports
outputs because that has a throughput benefit over traditional input queueing [5].

Ideally, the switch system should transfer the packets losslessly with minimum delay, while preserving the order in which they arrived. Obviously, the memory capacity in PRIZMA is limited and therefore has to be used efficiently. Hence, the internal switch memory is shared dynamically among all output ports.

Traffic is generated by a number of sources. In figure 2.1, only one source is connected to each input for reasons of simplicity. In reality, sources may be compound sources that consist of many sub-sources. The sources and destinations in the figure do not necessarily have to be the origins or final destinations of the packet flows. They might as well be other components of the network infrastructure.

The latest switch architecture, PRIZMA-T, will be discussed briefly in section 2.2. The very modular switch system can be used in various configurations. In order to provide some more background information, these will be discussed in more detail in section 2.3.

2.2 PRIZMA-T Architecture

The architecture of the PRIZMA-T switch is shown in figure 2.2. The input ports and output ports to the switch are shown on the left and right of the figure, respectively. The architecture consists of separate parts for data and control flow. In the following two sections, both parts will be described briefly.

2.2.1 Data Section

The data section, shown in the top half of figure 2.2 contains a shared memory that may hold a number of fixed size packets. The input routers are capable of forwarding a packet from any input port to any memory location. Similarly, the output selectors may
select any packet in memory for transmission from one of the output ports. At the input side, up to $N$ packets (one from each output) can be stored simultaneously and equally many can be retrieved from memory at the output side. The addresses for storing and retrieving packets from the shared memory are provided by the control section.

### 2.2.2 Control Section

The control section consists of a free cell queue and $N$ control queues; one for each output port. The former is merely a pool of free addresses, handled on a FIFO basis. A control queue holds the shared memory addresses of the packets that contend for its output. Note that the control section merely handles packet header information instead of entire packets. Each internal packet header contains a routing tag, which is used to route the proper free memory address to the right control queue. The memory address is also presented to the input routers, which use it to route the incoming packet to the correct free packet space in shared memory. Because the service mechanism of the control queues is FIFO, the order preserving property holds. When a packet’s address has progressed to the head of a control queue, it sets the router that belongs to its output port. Consecutively, the packet is transmitted, the address is removed from the control queue and added to the tail of the free memory address queue so that it can be reused.

A control queue is divided into a number of logical queues. This means that order preserving is done at the smaller scale of one logical queue. Furthermore, it enables packet selection based on a smaller grain, such as selection by packet priority. Selection
by priority is one of the most simple selection mechanism that distinguishes between logical queues. In case of high volumes of high-priority traffic, however, problems may arise. It may cause starvation of low-priority traffic and monopolization of the shared buffer by packets of a single priority. This will be discussed in further detail in section 2.4.

2.3 Modularity: PRIZMA-T Expansion Modes

Four different methods for performance expansion exist in the PRIZMA system. These expansion modes are independent of each other and can therefore be mixed to build systems of even better performance. Performance expansion involves combinations of multiple switch modules or combinations of physical ports on a single switch module. These expansion modes will be discussed in the four following sections.

2.3.1 Single-Stage Port Expansion

A single-stage $S \times S$ switch fabric can be built from basic $N \times N$ switch elements. Figure 2.3(a) shows an example of a $2N \times 2N$ fabric, which consists of four $N \times N$ elements. Generally, a single-stage $S \times S$ fabric consists of $(S/N)^2$ basic $N \times N$ switch elements. Single-stage expansion requires extra functionality at the ingress and egress of each switch element. Additional addressing is needed for distinguishing the switch elements that connect to the same ports of the $S \times S$ expanded fabric. Furthermore, extra arbitration is required on the output side for proper output port sharing of the expanded configuration.

2.3.2 Multistage Port Expansion

For very large fabric sizes, one may combine switch elements in a multistage configuration. Figure 2.3(b) shows a redundant three-stage topology which provides multiple paths from each input to each output. Therefore, the total load can be distributed over multiple paths. It is preferred to use as few stages as possible because packet delay is proportional to the number of stages. Moreover, load balancing becomes more complex as the number of stages increases. Note that large fabrics can still be built with a low number of stages because each switch element in the multistage configuration can be single-stage expanded if necessary.

2.3.3 Speed Expansion

Speed expansion, as shown in figure 2.3(c), yields expanded switch fabrics with port speeds that are multiples of the port speed of a single switch element. This is achieved by stacking multiple slave chips and have these controlled by a single master chip. The master switch element handles the packet header information and controls the data sections of all chips involved. The control sections of the slave chips are disabled. In the figure, incoming packets are split in four parts and fed to the master and slave switch
Figure 2.3: PRIZMA-T expansion modes
elements. Therefore, the port speed of the expanded configuration can be four times as high as the port speed of a single switch element.

2.3.4 Link Paralleling

Link paralleling is an expansion mode that does not involve the combination of switch elements. In this configuration, multiple single switch ports are combined to form an expanded port of greater capacity. Link 0 can be applied to the input ports as well as the output ports of a switch element.

2.4 PRIZMA-T Flow Control

2.4.1 Problem Context

The most important property of the PRIZMA switch is its losslessness. Once a packet from a so-called guaranteed traffic type has been sent from an input adapter to the switch, it may not be discarded inside the switch. The packet should appear at one of the switch’s outputs within a limited amount of time. Packet flows are statistically multiplexed [6] inside PRIZMA and this results in a certain probability of memory overflow, i.e., a probability of violating the losslessness property. This probability increases as the traffic sources become more active and more bursty in their transmission behavior. Since lossy switch behavior is not tolerable, the input flows should be regulated in such a way that packets with a delivery guarantee can always be delivered. There is a trade-off here between such a flow control scheme and the efficient use of bandwidth (throughput), resources (shared memory) and the delay properties of the switch system.

In-band Flow Control

The flow control scheme in the input adapters is based on status information from PRIZMA. This switch does not communicate this information directly to the input
I" input adapter (PRIZMA-T) output adapter

Figure 2.4: Physical paths of in-band flow control

 adapters, but embeds it in the packets that flow to the output adapters. Each output adapter notifies his peer input adapter of the current switch status as packets arrive there. Similarly, flow control information from the output adapters traverses through their peer input adapters before it reaches the switch. Figure 2.4 depicts the physical path of both in-band information flows. The advantage of in-band flow control is that the signal bandwidth of the external package pins is fully utilized instead of spoiled by a low-frequency signal such as flow control. However, a disadvantage is an increase in complexity and, in particular, the introduction of an extra latency in the communication of this important information. Latency plays a crucial role in the success of any flow control scheme.

Two Applications

The question about chip level flow control does not focus on one switch system application in particular. Two areas of investigation can be distinguished:

- Switching in telecommunication applications. The objective is to maximize traffic throughput, whereas switch latency is less important.

- Switching in microprocessor clusters. The system is generally offered a lower load than in case of telecommunication switching. Guarantees for low packet latency are crucial in this area.

Figure 2.5 on page 10 shows a typical graph that depicts the average packet delay versus the system's average input load and the two areas mentioned previously. Preferably we wish to use just one robust flow control mechanism for both areas.

Traffic Contracts

In the telecommunications area, we distinguish two types of traffic, namely guaranteed traffic and best-effort traffic. Under normal (non-malfunctioning) operation, the customer is 100% sure of delivery of the first type of traffic, i.e. packets will not be lost. Furthermore, its delivery rate and delay are guaranteed with a very high probability. A flow control scheme can support the delivery guarantee completely, but it can never fully guarantee the rate or delay because packet flows are statistically multiplexed in the switch, where resources are limited. Best-effort traffic is the type of traffic that may
be discarded at any stage in the system if traffic guarantees (for the guaranteed traffic type) in terms of delivery, rate or delay are threatened by the current congestion state. A traffic contract will prescribe a rate and delay for guaranteed traffic as well as a rate for best-effort traffic. The customer himself decides which part of his data traffic is of the first type and which is of the latter.

Figure 2.6 summarizes the subdivision of traffic into classes and types. Packets belong to a certain traffic class and each of the available classes provides a guaranteed and best-effort traffic type. This allows for a traffic weighting in terms of traffic classes as well as both traffic types.

2.4.2 Problem Definition

As described in section 1.2, one of our objectives is to build a model that is capable of supporting design decisions – decisions that are based on the evaluation of system properties. The system properties that should be guaranteed by flow control schemes are:

- Lossless switching with respect to guaranteed traffic.

- Availability of bandwidth. The switch system guarantees a certain minimum bandwidth that is available to each traffic class. Every traffic class can be subdivided into a guaranteed part and a best-effort part.

- Guaranteed access to switch resources. The possibility of identifying a packet by its traffic class should enable proper operation of the virtual lane concept. In this concept each traffic class is mapped to a virtual lane. The principle of a lane prescribes that traffic in a lane may not suffer from lack of resources because of traffic (congestion) conditions in another lane.

- A best-effort discard scheme that matches the three features mentioned above.
### 2.4 PRIZMA-T Flow Control

#### Traffic Classes

<table>
<thead>
<tr>
<th>Traffic Class</th>
<th>Guaranteed</th>
<th>Best-Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Class 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traffic Class 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traffic Class n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.6: Subdivision of traffic: guaranteed and best-effort traffic types per traffic class

#### 2.4.3 Previous Solution: Nested Thresholds

This section discusses an example of a flow control scheme used in an older generation of PRIZMA. It used to clarify a trade-off that comes with in-band flow control. The trade-off deals with the division of available bandwidth into data bandwidth and flow control bandwidth. Obviously, if the switch system is to operate efficiently in terms of data transport, the amount of (overhead) bandwidth reserved for flow control should be minimized. On the other hand, flow control information is important to make the system operate efficiently in the first place. Moreover, the processing of flow control information must be done as quickly as possible to keep extra latency to a minimum. This constraint limits the maximum level of complexity of the flow control information. A compromise must be found between those issues and this boils down to finding a flow control scheme that matches

\[
\max_{\text{control scheme}} \left( \frac{\text{performance}}{\text{overhead} \times \text{complexity}} \right)
\]

as closely as possible. Figure 2.7 on page 12 shows a number of logical queues that share an output port of the switch. Instead of communicating queue occupancy information of all logical queues, for example, it was decided to count only the number of packets contending for this output. The packet count is compared to a number of programmable thresholds and only the result of this comparison is sent in-band to the input adapters. Subsequently, the input adapters may deduct a measure for the current switch congestion state.

An example of a flow control scheme that uses these thresholds is the preemptive or nested priority scheme. In this setting, the input adapters stop transmitting packets of the lowest priority as soon as the lowest threshold is crossed (in upward direction). If the congestion state is such that the next threshold is also crossed, they will stop sending packets of the one-but-lowest priority as well, and so on. Similarly, the input adapters resume transmission of packets of a certain priority if the nested threshold for that priority is crossed in the other direction.

The disadvantage of this simple scheme is that it is prone to lead to starvation of the lower priorities in case a lot of higher-prioritized traffic is present. Although the
scheme itself is simple, it does not match the concept of weighted traffic classes and could therefore never yield a satisfying solution to the problem.

Note that we used the terms *priority* and *traffic class*. The difference between the two is the interpretation of the priority field in the packet header. As the name reveals, this field was initially used for storing the packet's priority in a scheme such as the one discussed above. In the context of the problem definition, it would be better to use the term *traffic class* since the term *priority* is confusing and might even exclude certain solutions.
This chapter describes POOSL (Parallel Object-Oriented Specification Language). First, it discusses the system level design methodology that gave rise to the development of the language. Secondly, it describes three structural entities (clusters, processes and data) that POOSL consists of. Next, we describe the syntax of process statements, data statements and data expressions. Finally, the two phase execution model is discussed, a model that is important to our modeling view that we present in chapter 4.

3.1 System Level Design Methodology behind POOSL.

Traditionally, system design is based on textual specification. From there, designers used to rush towards the implementation of the system. The result was, too often, that the incorrect systems were built or that developed systems did not work properly. As system complexity increases, this trial and error method is not recommendable. Instead, system design should be done by modeling and analysis. Designers should create formal executable system models that enable system validation and verification.

POOSL is a language that is developed for the latter approach. As opposed to a hardware description languages, such as VHDL, POOSL is a specification language. It is developed for specification of system level models at appropriate levels of abstraction instead of a detailed description of system architectures.

POOSL is equipped with a formal mathematical semantics. This allows for formal verification of models, in a qualitative as well as a quantitative manner. If one formalizes the properties that we wish to verify in a mathematical way, they can be checked automatically against the POOSL models.

POOSL is an expressively powerful language. With a relatively simple syntax, it enables specification of complex real-time models without the need for reinvention of concepts like time, concurrency and communication. POOSL is based on a timed version of the process algebra CCS (Calculus of Communicating Systems) [7]. It also incorporates the basic concepts of traditional object-oriented programming languages such as C++ and JAVA.
3.2 Structural Entities of POOSL

A POOSL specification consists of

- cluster classes,
- process classes and
- data classes.

The class of a process (a process class) specifies the behavior of a collection of similar process objects. The class of a cluster (a cluster class) defines the behavior and internal structure of a group of similar clusters. Data objects are also grouped in classes. A data class describes a collection of data objects with similar functionality. The individual objects of a data class are called data instances.

3.2.1 Process Objects

A POOSL specification consists of a fixed number of process objects, clusters of process objects and other clusters. Processes and clusters are statically connected by a topology of channels, through which they may communicate by exchanging messages. In POOSL, communication is based upon the pair-wise synchronous message passing mechanism of CCS.

The grain of parallelism in POOSL is the process object. Processes communicate by sending each other messages over channels. If a process wants to send a message, it explicitly states on which channel the message has to be sent. Processes also explicitly state when and from which channel they wish to receive messages. Immediately upon reception of a message, the sending process will continue its activity (it does not need to wait for a reply). If a process receives a message, it does not automatically execute a method like in traditional object-oriented languages. Neither does it automatically return a result to the sending process. If a result has to be returned, this has to be done by means of another communication between the two processes.

A process object can call and execute one of its methods. This does not require the reception of a message. Methods are comparable to procedures of imperative programming languages like C or PASCAL. Procedures of imperative programming languages, however, are expected to terminate and are therefore only able to express finite behavior. Methods in POOSL can be used to describe infinite behavior. Such infinite behavior can be specified by defining methods in a (mutually) tail-recursive manner.

Processes contain internal data in the form of data objects that are stored in so-called instance variables. Data objects are only accessible to the process object that possesses them, i.e. different processes do not share data. A process can use data objects by sending messages to them. When a process sends a message to a data object, its activities are suspended until the data object returns a result. Except for this result, data objects themselves cannot send messages to process objects.

When two processes communicate, a message and a (possibly empty) list of parameters are passed from one process to the other. The parameters refer to data objects that
are accessible to the sending process. Since processes cannot share data, it is not enough to pass references to the data objects like in traditional object-oriented languages. Instead, the objects themselves have to be passed. This means that new objects have to be created in the environment of the receiving process. These objects are so-called deep copies of the objects that are involved in the communication.

### 3.2.2 Clusters

A cluster may consist of processes and other clusters and behaves as an abstraction of these. Behavior cannot be specified in clusters because they are merely used to create a hierarchical structure of modules. Internal construction details of these modules are hidden.

### 3.2.3 Data Objects

In addition to process objects and clusters, POOSL also has data objects. A data object contains its own data and may perform operations upon this data. This data is stored in instance variables that contain (references to) other data objects. The variables of an object are encapsulated by that object; they are not directly accessible to other objects. These variables can only be read and modified by the object itself.

Data objects collaborate by sending each other messages. A message consists of a message name and zero or more parameters. A message can be regarded as a request to a data object; it is a request to perform one of its services. An object explicitly states to which object it wishes to send a message. If an object sends a message, it suspends all activity until the result of the message arrives. A data object that receives a message will execute a so-called data method. The name and parameters of this method correspond one-to-one with the name and parameters of the message that is received. A method implements one of the services of the object. The result of the execution of the method is returned to the sender. A method has access to all instance variables of the object. In addition, a method may contain local variables.

Besides data objects with instance variables, so-called primitive data objects exist. These are objects whose behavior is not defined by methods, but is primitive. Examples of primitive data objects are integers, reals, logical values and characters. Messages can be sent to these objects to perform standard operations. Logical values have methods to perform negation, logical ‘and’, and so on. Integers have methods to add, subtract, multiply, etc. One of the primitive data classes is a random generator. This class may be used within process classes or other data classes to provide probabilistic behavior.

### 3.3 Syntax of POOSL

Method bodies of process classes consist of process statements (PS). A summary of such statements is given in table A.1 in appendix A. Method bodies of data classes consist of data statements (DS) or data expressions (DE). These are described in tables A.2 and A.3.
3.4 Action Urgency: Two-Phase Execution Model

Like in many other process algebras, time behavior is based on a two phase execution model [8]. Figure 3.1 shows that the first phase is an asynchronous phase. In this phase, where time does not progress, all processes perform as many actions and communications as possible. The second phase is the synchronous phase in which time passes simultaneously for all processes. Time may pass by a delay statement, a communication that has no peer and a blocking guard. The actions in the asynchronous phase cannot be distinguished in time, although they do not occur simultaneously. These timeless actions are called urgent actions. Urgent actions are executed in non-deterministic order.
Chapter 4

General Switch Modeling Considerations

Prior to any modeling activity, it is important to shed a light on several general modeling issues and relate these to the system under investigation. These general issues should provide an insight on how the system can be modeled in the most suitable way, which greatly depends on the system properties that are to be evaluated. The most suitable modeling approach rests upon many aspects like model adequacy, clarity, efficiency, reusability and even convenience and intuitivity. Taking these aspects and their respective importance into account, the modeler should select a modeling approach from countless alternatives. This chapter describes general modeling issues like the relationship between models and architectures, a useful modeling view on real-time and functional behavior, communication and concurrency, decisions about the modeling of time and parametrizability. Furthermore, it aims to cover more practical switch-specific aspects, like the modeling of packet flows and collection and presentation of performance data. This chapter also presents a way to model slotted (periodic) packet flows, which greatly benefits from the expressive power of POOSL and is applicable to other communication environments as well.

4.1 Models and Architectures

In this project, we discovered that industry and academia tend to have different views on the relation between modeling and architecture development. In the academic world [1], emphasis is on specification of models for architectures, whereas in industry this view is usually reversed and people build a model of an architecture. The first kind is "modeling for design", the latter is "modeling for system property evaluation". An important observation in this project is that the PRIZMA-T system is not a design that is built from scratch, but is based on multiple preceding generations and therefore also faces compatibility requirements.

We discovered that there is not one right or wrong view among the two described above. Development of a new product will initially comprise of modeling for architectures\(^1\). The second type of modeling can be used to elaborate a more matured design,\(^1\) This does not mean that prescribed issues, such as dictated by previous generations or technological constraints are not taken into account.
such as PRIZMA-T. For this type of modeling, the modeler makes himself\textsuperscript{2} acquainted with a detailed specification after which proper abstraction is applied and a system model is built. The rewarding side to this \textit{roundtrip} into details is that it may yield an adequate and slender model of an architecture that can still be used to investigate future design solutions. An example of this benefit will be presented in chapter 6. As a conclusion, we state that both views are interchangeable and maximum gain is achieved by using the best of both.

### 4.2 Modeling View on Complex Real-Time and Functional Behavior

In POOSL, we distinguish three different classes of entities: cluster classes, process classes and data classes. Whereas the first type is merely an aid for adding hierarchy to a model, the latter two hold the actual specification. In process classes there is a notion of time, either by a specified delay, a communication statement or a guard. Data classes, however, are timeless. This means that data methods do not consume time and their result is returned instantly. If such actions should consume time in the model, this should be specified in the process class that owns the instance of the data class. The expressive power of data classes is in the area of the specification of complex functional behavior of a model. Although process classes are suitable for this task as well, they actual power lies in the specification of complex real-time and probabilistic behavior.

This most important difference between process classes and data classes, has great implications for our view on the model. As we prefer to specify the model in an elegant fashion, we decide to keep all process methods as small as possible. We intend to specify only the model's temporal behavior in process methods. All other (functional) behavior is kept behind the scenes in data classes. This modeling view simplifies discussions about temporal behavior, because of the relatively small size of process methods. Furthermore, participants of the discussion are not bothered by functional details at lower levels of abstraction.

### 4.3 Communication and Concurrency

Packets between PRIZMA and an adapter are transmitted during connected periodic intervals in time (so-called slotted transmission). If no packets are available for transmission, idle packets are transmitted to keep the link synchronized. Whereas \textit{communication} of all packets is synchronous, \textit{concurrency} of the \textit{input packet flows} is not. Packet flows at the ingress of PRIZMA come from different input adapters, hence these flows are concurrently asynchronous. As packet intervals are fixed, different packet flows show a fixed phase difference. Packets at the egress of the switch are transmitted simultaneously and are therefore concurrently synchronous. This is depicted in figure 4.1, where IA and OA denote input adapter and output adapter, respectively.

\textsuperscript{2} m/f
4.3 Communication and Concurrency

The question here is to decide whether it is necessary to model the per-link periodicity. The real-life packet flow coming from the sources is generally not periodic. To be able to answer questions about flow control in this system, it may be adequate to model fully aperiodic per-link communication for both the ingress and egress packet flows. The expressive power of POOSL supports such an abstract switch model. Since POOSL provides concurrent asynchrony for free, we will specify the ingress packet flows to be concurrently asynchronous, as described previously.

The question whether a fully asynchronous model is adequate or whether it is necessary to specify a (per-link) synchronous/slotted packet flow, cannot be answered beforehand. If simulation results point out that per-link synchrony is not relevant for the performance evaluation of flow control properties, the asynchronous model is adequate. Therefore we conclude that our first models should incorporate per-link synchrony. If flow control performance results prove its irrelevance, we may advance to a higher level of abstraction and continue with a more simple asynchronous model.

Before we are able to decide on that matter, we need to specify a synchronous model that is efficient in terms of simulation. Of course, it is possible to use a fixed delay loop in POOSL to assure synchrony. However, this approach will not yield an efficient execution of simulations with a low input load, like in the context of switching in microprocessor environments. In a real-life case of low input load, many idle packets will be transmitted between the adapters and PRIZMA. Before we start modeling straightforward delay loops, it may be rewarding to investigate some possibilities to build a more efficient and elegant model. These considerations will be discussed in section 4.4.2.
4.4 Modeling Time

Before we are able to discuss where time is to be specified in the processes of our model, we need to determine the grain of time that is required throughout the model. Since we aim to develop an elegant model and not over-specify it, we should take some distance from the hardware world where architectural components usually run by a common clock. This clock dictates time for an entire hardware component or larger parts of such a component and usually divides time in small slices – in the order of nanoseconds.

Nevertheless, we should not be distracted by this implementation issue and focus on the problem to be solved. Our problem is formulated around the concept of guaranteed bandwidth and guaranteed access but does not necessarily ask for answers in units of bits per second or bytes of available memory.

For modeling and simulation efficiency we should determine a time grain that is large as possible, under the condition that our model is still adequate. Hence, we would like to abstract from engineering units and deal only with measures of, for example, packets per unit time. Note that, at a later product development stage and higher (management) level of decision making, the performance results can always be calculated back to engineering units. Therefore we decide to set the time grain to one packet cycle, which is the (fixed) time it takes a packet to be transferred to or from PRIZMA. A larger time grain would prevent correct specification of switch behavior since statistical multiplexing in the shared memory of the switch is based on the grain of a packet.

4.4.1 Where to Model Time

Every instance that processes data packets should somehow consume time. POOSL provides several ways to specify time in a model. As described in chapter 3, time advances by

- an explicit delay statement,
- a pair-wise synchronous communication which cannot be completed because of the unavailability of a peer communication statement and
- a blocking guard.

Previous generations of PRIZMA had out-of-band flow control, which means that the actual switch chip(s) raised an external signal to notify the input adapters that data packet transmission should be stopped temporarily because of the switch congestion state. This mechanism is called back-pressure. Back-pressure is implicitly specified by a pair-wise synchronous transmission for which a matching reception statement is not available and can therefore be modeled with little effort in POOSL. This also holds for back-pressure between the output adapters and the switch chip(s).

In the case of in-band flow control, the processes in this system should (in principle) always be ready to receive packets since there is no immediate back-pressure, as explained in section 2.4. In the following we will speak of receptional willingness of a
process if it is ready to accept packets at all times. The receptional willingness property still does not force us to model time in one specific way. This can be explained if we consider another fundamental matter: the question whether the transmitting or receiving side of two communicating processes should make the decision which packets are to be communicated (see the example in figure 4.2). Basically, two solutions exist:

- The receiving end of two communicating processes is always willing to receive packets (and is also work-preserving\(^3\)), but it decides itself (by means of conditional reception) which packets should be communicated. Such a scheme can be based on a selection scheme that is embedded in a conditional reception. This solution proved to be very counter-intuitive and expensive (simulation-wise). The reason for that is that the selection scheme is based on in-band flow control information, which is available at the transmitting side (the input adapter in this example). Besides, modeling the short in-band flow control loop with latency inside PRIZMA would unnecessarily complicate switch specification. Apart from that, this awkward solution causes many pending pair-wise communications between sender and receiver that the simulator’s scheduler should match with proper reception statements. Additionally, it would be very expensive to evaluate complex conditional receptions.

- The transmitting end of the two communicating processes selects the packets that are to be sent, based on its selection scheme. Of course, this scheme will use in-band flow control information, as described previously. This solution is more intuitive, less expensive and divides the complexity of real-time behavior evenly among the processes.

In this case, the more intuitive solution happens to be the most suitable and therefore we choose to perform the packet selection in the transmitting process. As a consequence, time consumption should be modeled there as well. This can be explained as

\[^3\] The work-preserving property states that if a service is to be provided and servers or other resources are available, it is provided immediately without delay.
follows: the packet to be transmitted is selected from a number of logical queues that share one output. Other non-selected packets should wait for another selection round and therefore be delayed for at least an extra packet cycle. This means that at least some time must be consumed in the transmitting process.

In the light of elegance, we therefore decide to model all time consumption only in transmitting processes. Moreover, this decision matches very well with the requirement of receptional willingness. Since all communication in POOSL is pair-wise synchronous, it is very convenient to specify a receiving process that does not consume time by means of a delay or guard statement. If a tail-recursive receiving process performs all its actions in zero-time, it never implicitly back-pressures the sending process and is therefore always willing to receive. (Note that a receiving process can both consume time and satisfy receptional willingness under only one condition, namely in the case that the transmitting process has nothing to send).

We conclude with a heuristic that we used in the modeling of receiving and transmitting processes in this system model of PRIZMA. The heuristic is that for performance analysis in a packet forwarding and switching system of such uniform structure, we only need to model time consumption of packets in only one location of the process class, i.e. either in the receiving or transmitting process, but not in both. The use of this heuristic generally leads to an elegant specification and facilitates discussions about real-time behavior. The concepts that were discussed in this section will be made more concrete by the POOSL examples in chapter 6.

4.4.2 Packet Flow Modeling

This section discusses two approaches to packet flow modeling in the PRIZMA-T system: rest time calculation and decision moment calculation. Both show slotted transmission of packets from a queue and are based on abstraction from idle packets. Of course, this abstraction is not valid for all questions in general, but will prove very useful for questions about performance at the system level.

Rest Time Calculation

In this simple example we consider a process that contains multiple input ports and one output port, connected by a single queue. Every packet that enters this process through one of its input ports is put in this queue and must be transmitted from the output port after some time. If no input actions take place and the queue is empty, the process stalls (by means of a guard). If multiple input actions occur within one packet cycle, the packets are queued on a FIFO basis.

Figure 4.3 shows the communication actions of this process. The time line preceded by \textit{in} and \textit{out} shows the input and output actions, respectively. The vertical arrows denote the moments at which communications take place. Input actions may occur randomly in time, whereas output actions are to be scheduled at fixed (periodic) instants. The principle of rest time calculation is only to act when a relevant communication

\[4\text{ i.e. one or multiple times within the same process method of the class}\]
4.4 Modeling Time

should take place. That is, the process should only act if a data packet is to be received or transmitted, since these communications are relevant for performance evaluation, as opposed to idle packets. The communication of idle packets in real life is, in fact, merely an implementation decision; a decision that our system model can do without.

Assuming that some initial "synchronization instant" (say, $\tau$) has been agreed upon, output actions are scheduled at instants

$$t_{out,k} = \tau + kp,$$

with $k \in \mathbb{N}$ and $p$ the time span of a packet cycle. If packet $B$, for instance, enters an empty process queue at an instant equal to $t_{out,k}$, $k \notin \mathbb{N}$, it is delayed for a time $\Delta_B$, after which it is transmitted. In the example, this delay is a simple difference of the transmission time $t_{out,2}$ and arrival time $t_B$. Packet $C$ enters a non-empty process queue and therefore has to be transmitted after a delay of $\Delta_C$. A queue's non-emptiness is evaluated in a guard at the start of the transmitting process method. In this example, this means that the queue's occupancy state is not evaluated until packet $B$ has been transmitted. At this instant, the proper delay($\Delta_C$) for the packet at the head of the queue is calculated. Subsequently, the process is delayed for $\Delta_C$.

It is important to note that the delay time of a packet can be calculated right from the start of the process method. Since this process contains a single queue, a decision need not be taken and therefore the calculation can be performed immediately upon a non-empty queue state. However, problems arise if more elaborate queueing schemes are used. This problem is tackled in the following.

**Decision Moment Calculation**

If incoming packets of different priorities or traffic classes simultaneously enter multiple logical queues within the same output queue, the process needs to decide which
packet is to be sent first from this queue (through a common output port). The moment of decision is important for proper operation of any selection mechanism. If the decision is made upon entrance of a packet, proper operation cannot be guaranteed in general. The reason for this is that another packet—a better candidate—may enter the queue within one packet cycle of the arrival of its predecessor. The better candidate should also be taken into account when making the transmission decision. The previously described rest time calculation does not support this mechanism because a delay is calculated for each individual packet at the head of a queue. In other words, it does not support the sharing of one output by multiple (logical) queues in a temporal sense.

Therefore we propose a slightly different approach as opposed to the rest time calculation. The principle of decision moment calculation is shown in figure 4.4. Similar to figure 4.3, it shows the input and output actions on two (identical) time lines. The bottom time line shows periodic instants at which transmission decisions are made. We specify the model in such a way that no other process may remove packets from this queue, since each parallel transmission process has the exclusive rights to operate on its queue. Assume, in this example, that all arriving packets are destined for just one output. Note that the bottom part of figure 4.4 does not show a snapshot of the logical queues at some point in time. It is just there to illustrate the classification of prioritized
packets in this example.

If a packet (packet $A$, for example) enters an empty queue, a guard is opened and a delay for the next decision moment is calculated ($\Delta_1$). In case the queue is part of the switch, the packet needs to reside in the queue for a minimum time span of one packet cycle. Therefore, it is delayed until (at least) the second transmission instant after its arrival. For packet $A$, this is at $t = \tau + (k + 1)p$. Note that these instants are merely decision moments. This is illustrated by the reception and transmission of packets $B$ through $D$. Packet $B$ arrives at $t_B$ and a decision moment is scheduled after a delay of $\Delta_2$. In the mean time, packets of higher priority (packets $C$ and $D$) enter the queue. At $t = \tau + (k + 3)p$, packets $B$ and $C$ will be taken into consideration by the selection procedure. Packet $D$ will be ignored by the selection procedure because it has been in the queue too briefly. In this example, the procedure selects packet $C$ for transmission. This means that packet $B$ will remain in the queue until (at least) the next decision moment. At $t = \tau + (k + 3)p$, the guard of the transmission process is still open because the queue is not empty. Hence, a delay time $\Delta_3 = p$ is calculated. Generally, all packets that are not selected will compete in another selection procedure, which is scheduled for the current (decision) instant plus a delay of exactly one packet cycle to ensure synchrony.

In order to decide which packet has been in the queue for a sufficient period of time, the selection procedure at the head of the output queue needs to take the current simulation time into consideration. This time is compared to the proper time stamp field in the packets that are under investigation. Note that this is only required for the proper specification of system behavior under the abstraction from idle packets. The inclusion of time in the selection procedure is not required in an implementation of the switch system.

The decision moment calculation principle guarantees slotted transmission without the synchrony of a fixed delay loop. Furthermore, because of the guard construction, the existence of a best candidate packet is guaranteed, which simplifies the specification of the decision moment calculation in POOSL. We may conclude that decision moment calculation is a simple and efficient way to model slotted packet flows, regardless of which packet selection mechanism is used.

Future system analysis may show that fully asynchronous models are adequate to answer our questions on system performance. However, decision moment calculation is a convenient alternative in cases where slotted communication may not be abstracted from. For more details about this principle, refer to chapter 6 where an example of decision moment calculation is presented.

4.5 Parametrizability

Parametrizability of a system model is important for several reasons. First, it supports flexibility and modularity. Parametrizability enables analysis of a single system with different settings, e.g. simulations of a switch with a different number of ports or traffic classes. In this context, it may also speed up the modeling phase of a future system of
larger size. Secondly, in the scope of PRIZMA, parametrizability may be used to model modularity. A speed-expanded, single stage port-expanded or link paralleled system can be specified within one single "switch" process that behaves like the expanded system. The advantage of this modeling approach adds up to the flexibility of parametrization of the number of ports.

Secondly, parametrization yields modeling convenience and clarity. Often, in the exploratory modeling phase, the modeler does not need to run simulations of the full system to get acquainted with the problem. It is much more convenient to explore a switch system with a small (2, 4, ... ) number of ports and small number of supported traffic classes than a full-blown system.

Simulation of such a large system model is slower than the unparametrized model and presentation and analysis of results may take unnecessarily much effort. This relates to a drawback of parametrizability, which becomes more clear after the exploratory modeling phase is completed. When a model is finished and it is only used for complete system simulation, the flexibility and convenience of parametrizability is paid back in terms of simulator performance. In the case of POOSL, this decrease in performance may be caused by parametrization of the number of ports of a process. In that case, only one communication channel is needed between the senders of the message and the receiving process(es). In some cases, the need for conditional receptions or guards that discriminate between senders in parallel input processes may slow down the simulation.

4.6 Acquisition and Presentation of Performance Data

Because performance metrics are required that involve time, our model should be able to use the simulation time during execution. The POOSL timestamp process statement provides this functionality. In fact, without this statement it would be impossible to base our model on the important principles of rest time and decision moment calculation, as described in section 4.4.2.

4.6.1 Presentation of Results

Currently, SHEsIM itself does not provide a comfortable environment for system level performance analysis. For now, performance analyses will mainly be performed off-line – by scripts that operate on data generated by the model. Since SHEsIM does not offer the possibility yet to make plots of a performance metric, all data that is required for such a plot is written to simulation log files. Performance analysis methods that prove to be useful, can be made available in the simulator for more rapid on-line analysis in the future.

As will be shown in chapter 5, the Markov Chain Monte Carlo [9] statistical analysis method will be performed by the model. This enables the model to take decisions on the results of this analysis, such as stopping a simulation when a required confidence level is met. Future plans exist for removal of such decisions from model itself and embed these into the simulator. The current developmental state of SHEsIM does not
support such a segregation yet and therefore we will have to integrate this functionality with the actual model specification. Note that this small "pollution" of the POOSL code only affects the elegance of the specification; its functional possibilities remain equally powerful.

4.6.2 Acquisition of Data

Until the separation of model specification and on-line performance analysis is realized, we need a temporary solution for data collection. This can be carried out in several ways. We could, for example, specify an auxiliary process class that connects to all relevant system processes. In that case, all system processes could communicate with such an observing process instance and data collection would be performed centrally. A disadvantage of this solution is that it is not structure-independent. For each process class that communicates with the observer, we need a channel and a port. Moreover, the biggest concern is the integration of these communication statements with the real-time behavior of the system. This solution results in a more widespread pollution of POOSL code.

Indeed, a better solution exists in the context of a switch and its environment; one that performs data acquisition in timeless data classes. As will be shown in chapter 6, data collection can be done in a "sink" data class, such as the destination of the packet flows. The object-oriented nature of POOSL enables storage of measured values in data objects, such as packets. Simulation time stamps or memory occupancy values could, for example, be stored in data packets as they travel through the system. Each process instance a packet visits, may store some data in the packet. These values can be analyzed or logged at the packet flow's final destination. An advantage of this solution is that it causes only very minor pollution of existing POOSL code.

In the exploratory modeling phase, other means of data collection may be added to the model. An example is an overflow buffer for packets that are discarded in the switch because of incorrect model specification or a bad flow control mechanism. This overflow buffer is, of course, not present in PRIZMA but may be helpful for the modeler in this phase of the project.

Finally, we mention that future analysis may involve co-compilation of certain specific questions with the abstract system model. As a result, an executable model will be able to answer these questions directly, without the need for off-line analysis.

4.7 Conclusions

Even from a third-generation product, such as PRIZMA-T, it should be possible to specify a model of an architecture that does not only support performance analysis, but that can also be used as a model for future architectures in this context. This latter feature depends on the abstraction level applied to the initial model of the architecture.

Switch modeling comprehension and discussions about temporal behavior can be facilitated by a modeling view that prescribes to model all real-time behavior in POOSL
process classes and timeless functional behavior in data classes as much as possible.

In order to speed up the creation of an abstract system model, initial system architecture specifications should contain a section on communication and concurrency. Although the modeler should still spend time on the details of that document, clearly describing these two areas will facilitate modeling greatly.

It is not necessary use a model time grain that is smaller than a packet cycle\(^5\). In addition, we may abstract from engineering units and calculate in packets per unit time. Taking POOSL's pair-wise synchronous communication into account, it is recommendable to model packet communication decisions on the transmitting side of two communicating processes and specify a timeless receiving process. The expressive power of POOSL enables efficient modeling of slotted packet flows without the need for a fixed periodic delay loop.

We recommend specification of a fully parametrizable model. This choice is based on the benefits of parametrizability in the areas of flexibility, modularity, clarity and convenience. In the exploratory modeling phase it is better to use a fully parametrizable model, whereas in the performance analysis phase, simulator performance may suffer from parametrizability.

The POOSL timestamp process statement is \textit{required} for the type of performance analysis that should be carried out. Because POOSL was not designed as a language for computationally expensive performance analysis and SHESIM does not support convenient presentation of performance analysis results, we choose to have those tasks performed off-line. The latter decision also matches our modeling view and prevents pollution of POOSL code with functionality that should be separated from model specification in the first place.

\footnote{A packet cycle is the time it takes a packet to be received by or transmitted from PRIZMA.}
Chapter 5

Performance Analysis

This chapter discusses which quantitative properties should be evaluated by our model, how traffic measurements should be classified and what simulation results should be presented. It also considers the advantages and disadvantages of analytical and empirical analysis. The Markov Chain Monte Carlo statistical analysis method is described and an example of the application of this method is given. Finally, we discuss the suitability of the method for all performance metrics that should be evaluated.

5.1 Definitions and Analysis Context

5.1.1 Definitions

The flow control problem definition, as described in section 2.4.2, states a number of features that should be guaranteed by a flow control scheme. In order to investigate the performance of flow control mechanisms we should choose the performance metrics needed for such an analysis. These will be defined in the following.

Performance Properties

Load/Throughput: Both load and throughput are measures of traffic activity. Load is determined by the sources and is measured at the input of the switch system. Throughput is measured at the output and is dependent on how well the switch can cope with congestion. It is a measure of packets per unit time in the connection under investigation. Therefore, utilization of bandwidth can be obtained through measurement of load or throughput.

Delay/Jitter: Although the problem definition does not explicitly state requirements on delay and jitter (variance of delay), these measures will be necessary for the comparison of the service that is given to different traffic classes. Furthermore, these metrics contribute to better comprehension of the switch system as well as validation of the modeling process.
Memory Occupancy: For the evaluation of a fair distribution of switch resources, it is necessary to measure fill levels of the shared memory and queues. This metric will provide information on efficiency and fairness of memory usage.

Traffic Classification

Our problem definition states properties that are to be evaluated for traffic classes. Therefore, we should at least distinguish traffic by the traffic class it belongs to. Besides that, it is important to obtain a notion of monopolization of switch resources. Without any form of flow control the shared memory of PRIZMA-T can be monopolized by traffic that belongs to a single traffic class or traffic that is destined for a single output port (a so-called output class). Similarly, a control queue for a certain output port can be monopolized by a single traffic class. In order to become aware of these forms of monopolization, it is required to distinguish traffic by its traffic class as well as its output class. We conclude that, for clear comprehension of the switching process under various conditions, we should classify traffic by

- input class,
- output class,
- traffic class,
- traffic type (guaranteed or best-effort) and
- general class (no classification).

The latter can be used to verify general load, throughput and delay conditions.

Metrics and Units

As mentioned before, we attempt to use as little engineering units as possible. Performance metrics should not be given in megabits, bytes or microseconds, but rather in percentages\(^1\), packets and packet cycles. This facilitates the comparison of switch systems of different size and capacity. Table 5.1 summarizes the metrics to be analyzed and the units in which these will be expressed.

Graphical Representation

For proper comprehension and performance evaluation, a number of graphical representations of the performance metrics should be given. These are

- switch throughput versus input load,
- packet delay versus input load,

\(^1\) or as a fraction in the interval \([0, 1]\)
5.1 Definitions and Analysis Context

Table 5.1: Metrics and units

<table>
<thead>
<tr>
<th>Metric</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>load/throughput</td>
<td>percentages of line capacity</td>
</tr>
<tr>
<td>delay</td>
<td>packet cycles</td>
</tr>
<tr>
<td>jitter</td>
<td>packet cycles</td>
</tr>
<tr>
<td>memory and queue fill levels</td>
<td>packets</td>
</tr>
</tbody>
</table>

- memory and queue occupancy versus input load,
- probability distribution of packet delay,
- probability distribution of memory or queue occupancy and
- all of the required performance metrics versus time.

The latter can be used for system comprehension in the exploratory modeling phase.

5.1.2 Analytical versus Empirical Evaluation

Any model that is specified in POOSL implicitly defines a Markov chain because of its probabilistic behavior (figure 5.1). Each scheduler step of a simulator that interprets POOSL code can be regarded as a state transition in this Markov chain. For performance analysis, one could calculate the state equilibria. If all state equilibria are known, we are able to determine all performance metrics possible (given that the model is adequately specified for such a metric).

However, the size of the Markov chain greatly depends on the complexity of the system. For simple systems, a rather small Markov chain is defined and performance can be determined analytically. With the increase of system (model) size, the number of states of the Markov chain explodes and we are no longer able to calculate state equilibria within an acceptable or even finite amount of time. Therefore, analytical evaluation

Figure 5.1: A model specified in POOSL implicitly defines a Markov chain
is not an option with current constraints on product development time. Independent of the state of technology, it is not possible to develop state-of-the-art systems using state-of-the-art processing power if performance evaluation is done in an analytical fashion. Only by thorough abstraction one might carry out performance evaluation analytically, but this will generally not lead to adequate models.

Given the constraints on product development time, we choose the empirical approach to performance evaluation. We will simulate one or multiple traces of the abstract system model and use statistical methods for performance analysis. If the simulation is executed for a sufficient amount of time, it will be possible to make an estimation of performance properties. Note that it is not possible to retrieve the exact values as is the case with the analytical approach. However, statistical methods allow us to specify an (approximate) confidence level for the accuracy of the result. In practice, this confidence level is set by the modeler or developer and will be chosen in the vicinity of 100%. Section 5.2 describes a statistical analysis method that is suitable for this approach.

5.2 Markov Chain Monte Carlo Analysis Method

The Markov Chain Monte Carlo method [9] comprises of a synthesis and an analysis part. The former part deals with the empirical construction of a Markov chain from the requirements for state equilibria. The latter part tries to retrieve these equilibria (or functions thereof) by simulating an execution trace in the chain.

Figure 5.1 depicts a Markov chain. The states in this diagram are steps of the simulator's scheduler. The black dots are states that contribute to the performance of the switch system. Such a state could, for example, represent the communication of a data packet—a packet that is relevant to our performance questions. The gray-colored dots represent scheduler steps that are not directly relevant to these questions. The gray states could represent the non-deterministic urgent actions in POOSL (see also chapter 3). Generally, the black states will be separated by many gray states. This is relevant because it causes the occurrence of black states to be rather independent. The independent occurrence of Markov states that are relevant to performance questions allows us to use the central limit theorem for our analysis.

**Theorem (Central Limit Theorem) [10] [11]**

Let \( X_1, \ldots, X_n, \ldots \) be independent random variables that have the same distribution function and therefore the same mean \( \mu \) and the same variance \( \sigma^2 \). Let \( Y_n = X_1 + \cdots + X_n \) and

\[
S_n = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (X_i - \mu)^2},
\]

(5.1)
5.2 Markov Chain Monte Carlo Analysis Method

which is an estimate of the standard deviation. Then the random variable

\[ Z_n = \frac{Y_n - n\mu}{S_n \sqrt{n}} \]  

(5.2)

is asymptotically normal with mean 0 and variance 1; that is, the distribution function \( F_n(x) \) of \( Z_n \) satisfies

\[ \lim_{n \to \infty} F_n(x) = \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-u^2/2} du. \]  

(5.3)

If \( X_1, \ldots, X_n \) are independent random variables with the same mean \( \mu \) and the same variance \( \sigma^2 \), then their sum \( X = X_1 + \cdots + X_n \) has the following properties.

1. \( X \) has the mean \( n\mu \) and the variance \( n\sigma^2 \).

2. If those variables are normal, then \( X \) is normal.

If those variables are not normal, then the second property fails to hold, but if \( n \) is large, then \( X \) is approximately normal and this Justifies the application of methods for the normal distribution to other distributions, but in such a case we have to use large samples.

Note that this theorem is even valid if the \( X_i \) are not independent. However, in that case, the unknown \( \sigma \) can no longer be estimated by the standard variance estimate in equation 5.1. In the Markov Chain Monte Carlo technique, other estimates, such as the batch means, are used instead. In this thesis, however, we will not use this and assume independence of the \( X_i \).

We will now show an example of the estimation of the average throughput of packet traffic on a channel. The throughput will be derived from the inter-arrival times (iat) of the packets on the channel. The throughput and delay metrics can be estimated in a similar way.

Example of the application of the Markov Chain Monte Carlo method [12]

Let \( \tau_i \) be the arrival time of a packet on the channel. The inter-arrival time between packet \( (i) \) and packet \( (i-1) \) is

\[ iat_i = \tau_i - \tau_{i-1}. \]  

(5.4)

We would like to estimate the real average inter-arrival time \( \mu \) as could be obtained through extensive analytical evaluation. We define a point estimate (after simulation of \( n \) packets) of the average inter-arrival time \( \overline{iat}_n \).

point estimate: \( \overline{iat}_n = \frac{1}{n} \sum_{i=1}^{n} iat_i. \)  

(5.5)
We want to obtain an estimation (again after simulating $n$ packets) for the average throughput ($\text{throughput}$), for which holds

$$\lim_{n \to \infty} \text{throughput}_n = \lim_{n \to \infty} \frac{i \text{at}_n - 1}{n}.$$  \tag{5.6}

If we want to know the accuracy of the point estimate in equation 5.5, we also need the variance of the inter-arrival time ($\sigma^2_{\text{iat}_n}$).

$$\text{point estimate: } \sigma^2_{\text{iat}_n} = \frac{1}{n-1} \sum_{i=1}^{n} (i \text{at}_i - \bar{i \text{at}_n}).$$  \tag{5.7}

From the central limit theorem we may now derive that

$$\lim_{n \to \infty} \frac{\sqrt{n}(\bar{i \text{at}_n} - \mu)}{\sigma_{\text{iat}_n}} = N(0, 1).$$  \tag{5.8}

In order to calculate a confidence interval around our estimation of the average inter-arrival time, we should choose a confidence level $\gamma$ (a value in the interval $[0, 1]$). The confidence interval is symmetric around the (estimation of) the mean. For the $N(0, 1)$ distribution, this boils down to finding the borders of the confidence interval $[-c, c]$ around 0. We should determine $c$ such that

$$P(-c \leq N(0, 1) \leq c) = \gamma.$$  \tag{5.9}

Combination of equations 5.8 and 5.9 yields

$$\lim_{n \to \infty} P(\mu \in \left[ \frac{-c_{\text{iat}_n}}{\sqrt{n}}, \frac{c_{\text{iat}_n}}{\sqrt{n}} \right]) = \gamma.$$  \tag{5.10}

We have now obtained an estimated interval that contains the real average inter-arrival time $\mu$ with an approximate probability $\gamma$. The estimate of the average throughput is as shown in equation 5.6. The left and right margin of the confidence interval of the average throughput are also inversely proportional to the right and left margin of the interval of the inter-arrival time, respectively.

The Markov Chain Monte Carlo method may also be applied to performance measures such as throughput and delay. Although jitter itself may be measured as the variance of the delay, more research is required for the estimation of a confidence interval for jitter.

The estimation of the fill levels of memory and queues is not as straightforward as one might think. The interest for this fill level is the estimation of a resource overflow probability. So in fact, we should estimate the size of the resource that leads to an overflow probability of $(1 - \gamma)$. First we should decide what an average fill level actually is. There are multiple ways of measuring an average fill level. The first method is periodic inspection of the fill level with a sufficiently small time granularity. The second method is only to react to changes in the fill level and apply some kind of "time-weighting". In the latter case, the Monte Carlo method should be applied to a metric
that depends on two stochastic variables, namely a fill level and a time span. Of course, the expressive power of POOSL calls for the time-weighting method. The method that was described in this section cannot be applied directly to the problem of memory and queue occupancy. More research is required in this area.

As the number of samples (simulated events/packets) increases, the confidence interval becomes more narrow around the point estimate of the mean, provided that the confidence level $\gamma$ is fixed during simulation. When the desired confidence level for a certain performance property is achieved, the model itself may decide to stop the simulation. This means that the modeler no longer has to guess or iteratively determine the number of samples that is deemed "sufficient". As an example, the model may decide to stop when the width of the confidence interval for the bandwidth share of a certain traffic class is less than 2.5% of the measured mean load for that class, with a confidence of 99%.

Note that this method seamlessly matches with traffic contract guarantees, as discussed in section 2.4. The probability of rate and delay guarantees of a certain traffic class may be set to a value that approaches 100%. Another advantage of this method is that we are able to obtain a confidence interval with only one execution trace that is exactly long enough to provide us with an answer on performance questions.

An issue that requires more research in the context of the switch system is the definition of a "warm-up" period in which no performance analysis is done until the behavior of the sources has converged close enough to its intended (stable) behavior. If we specify sources with a rather low traffic activity but high burstiness, it will take a long time before they settle close to their intended values. Fortunately, the simulation will not stop (with very high probability) until the required accuracy is met. In this way, modeling results cannot be influenced by a (false) simulation length that is determined by the modeler.

Chapter 6 will describe how the the Markov Chain Monte Carlo method is applied in the model.

5.3 Conclusions

The flow control problem definition requires several performance properties to be evaluated. These are load, throughput, delay, jitter and memory occupancy. Furthermore, we need to subdivide measurements on traffic into different classes to be able to answer the questions for these classes. Classification of traffic measurements will also facilitate comprehension of monopolizing behavior of traffic classes. The classes we distinguish are input classes, output classes, traffic classes, traffic types (see also section 2.4.2) and a general class.

Simulation results will not be presented in units of megabits, bytes and microseconds, but rather in percentages, packets and packet cycles. As mentioned before, graphical representation is not yet possible in SHESIM. Therefore we decide to let the model generate performance data that is suitable for off-line conversion to a graphical representation.
The analysis part of the statistical Markov Chain Monte Carlo method can be applied to the Markov chain that is implicitly defined by POOSL specifications. This empirical method can be used to obtain a confidence interval for the estimation of performance metrics with only one execution trace of the model. The method will be applied on-line and will therefore be able to stop the simulation if the required accuracy is met.

Additional research is required on the application of the Markov Chain Monte Carlo method for the determination of confidence intervals for jitter (variance of delay) and memory and queue fill levels.
Chapter 6

An Abstract System Model

This chapter discusses the path that is taken to construct an abstract system model of PRIZMA-T and its environment. We discuss how an adequate model can be specified from the information that is given in a product's detailed architecture specification. This is illustrated by the many abstractions we were allowed to make in order to specify a compact model that is able to provide an answer for the questions on flow control. Abstractions can be divided into intuitive abstractions, abstractions from architecture structure and communication and concurrency abstractions. Next, we describe where the model performs the actual performance analysis. Finally, each process in the model is discussed briefly.

6.1 Introduction

It is not for arbitrary reasons that this chapter is called 'An Abstract System Model'. In our general switch modeling considerations in chapter 4 we already mentioned that the modeler may choose a modeling approach from countless alternatives. The final choice is determined iteratively during the exploratory modeling phase and depends on the questions that are to be answered by the model. It has proved helpful to think in terms of non-redundant adequacy and focus solely on the relevance of system aspects in relation to those questions.

Prior to the actual modeling phase but after the general modeling considerations we focused on the architecture specification of IBM's PRIZMA-T. We found that this product specification at the architecture level contains quite some details - sometimes even implementation issues. This is not a negative observation. As a matter of fact, it is very important for a modeler to spend some time on the details and a priori prescriptions of the product that is to be modeled. Only in this way, he will get acquainted with the actual problem and is able to verify the validity of his abstractions.

We found that the architecture specification mainly discusses 'how' the system should be developed, instead of 'what' it is supposed to do. It is very important to abstract from the 'how' and model the behavior of 'what', if it were only to prevent overlooking conceptual solutions. The modus operandi of our modeling activities is a constant awareness of the difference between 'how' and 'what' and evaluation of the relevance of all issues that cover the path towards an abstract system model.
In the following section we attempt to convey the abstract modeling approach in the areas of system structure, communication, concurrency and, more specifically, flow control.

6.2 Abstractions

6.2.1 Intuitive Abstractions

Intuitive abstractions one can easily make in the context of performance analysis are the choices of time and data granularity. We have already discussed the issue of time granularity in chapter 4 and came to the conclusion that it is not necessary to specify a clock-cycle true model. The time grain in our model is a packet cycle. The second intuitive abstraction is data granularity. From the perspective of our problem definition, it is not relevant to specify packets at a detailed level. Therefore, we will not model the contents of PRIZMA-T packets, except for a number of tags that we need to do the performance analysis. These tags are the input port number, output port number, traffic class, traffic type, several time stamps and the packet’s sequence number.

6.2.2 Architectural Structure Abstractions

Although we started building a model of an architecture description, a one-to-one match of model and architecture structure is not required. We used this freedom to build a compact model. The system overview in figure 2.1 shows $N$ separate links between the adapters and PRIZMA-T. Because these links are separate, we conclude that they are independent. For reasons of parametrizability we may model these independent links as one channel in POOSL. This channel serves logically and independently multiplexed packet flows.

Another abstraction of structure that is in line with the parametrizability concept is the abstraction from $N$ separate input or output adapters. The POOSL process statement for parallel composition can be employed to specify the adapters as independent parallel processes within one process object. Note that the modeler should remind that adapters do not have knowledge of their neighbors.

In our first models we choose to model the shared memory of PRIZMA-T as a fully shared (unpartitioned) storage for packets. In doing so, we may abstract from the structure of a separate data section control section (see section 2.2). We only model the shared memory as a counter of the number of packets that is inside the switch. In our model, the data section and control section have merged to a single section that stores the actual data packets in FIFO order. Our modeled switch will put packets that contend for a certain output in a FIFO queue, in a similar way as memory addresses are put in control queues in the real-life case.

Although our modeling attempts mainly focus on a single-switch system, we do have some modeling ideas for the modeling of PRIZMA-T expansion modes, which were discussed in section 2.3. In that case we recommend abstraction from structure on a “per stage” basis; that is, one switch process can behave like a single stage of parallel
switches. This means that we would need as many switch process objects as stages in the case of multistage port expanded configuration. This abstraction requires a multiple administration within the switch process, but that should not be a problem for reasons of uniformity.

6.2.3 Communication Abstractions

As a part of our general modeling consideration we have already discussed the modeling of packet flows. Two modeling techniques were presented that do not require specification of a fixed periodicity as one would find in hardware description languages. Our modeling is based on a lazy model that only acts when necessary while preserving the behavior of slotted communication. In the same fashion, we will communicate flow control information. Furthermore, in our model only the changes in flow control information are communicated. This valid abstraction makes our model more efficient in terms of simulation speed.

All values that represent time in our model are integer instances. The simulation time in SHESIM is measured in "ticks", which are real-valued. We chose not to abstract from slotted packet communication and therefore did not specify a fully asynchronous model. After thorough simulation and comprehension of system behavior, non-redundant adequacy may still allow us to do so. This means, for now, that we should also model phase differences of the input flows of the input adapters and PRIZMA-T. Since packet cycles were chosen to be an integer number of simulation ticks, the phase difference grain is determined to be just one (integer) tick. This is a valid abstraction because (theoretically) an infinitesimal phase difference grain can be modeled. We can achieve this by defining an integer packet cycle that approaches an infinite number of simulation ticks while keeping the phase difference grain as it is.

6.2.4 Concurrency Abstractions

Figure 6.1 shows one of the abstractions we made in the area of concurrency. The figure depicts a timeless receiving process that satisfies receptional willingness (see section 4.4) and parallel output processes that are activated by non-empty queues. This abstraction matches with the decision of modeling time only in the transmitting process. Packets that enter the process do not consume time until they take part in the packet selection procedure in one of the output processes.

The specification of a timeless receiving process also proved to be advantageous in another way. Simply because the receiving process does not consume time and is always willing, it may accept every packet at every time instant. Since all packets enter the process through one channel, we do not even have to use the expressive conditional reception in POOSL. If \( N \) logically multiplexed packet flows were to communicate with an equal number of parallel receiving processes, our model would be more expensive in terms of simulation. The reason for that is the non-deterministic peer matching

---

\[1\] Note that the diagram in figure 6.1 could either be an input adapter, output adapter or PRIZMA-T!
An Abstract System Model

Figure 6.1: Timeless receiving process and triggered output processes

of pair-wise communication in POOSL, of which simulation costs rise with the number of communicating parties (ports) in the model.

We conclude with a few words on atomicy. Atomic statements in POOSL are crucial for valid specification of our model. They are mainly used in combination with communication statements. When a packet enters or leaves a queue because of a communication, queue status information should be updated instantly. If other urgent actions were able to alter the queues status we would have incorrectly specified system behavior. Process statements cannot be used in atomic statements because they may consume time. If such atomicy is crucial for proper behavioral modeling, the functionality of the process method should be split into a non-time consuming part and a time-conscious part. The first part can now be put into a data class and included into the atomic statement.

6.2.5 Flow Control Related Abstractions

We observed earlier that the real system structure does not need to match with the behavioral structure in the model, for the questions that were posed. Good reasons for not letting those structures match are that it would make the model more compact, discussible or more efficient in execution. Chapter 4 already suggested a packet flow modeling technique that abstracts from idle packets. The question here is whether the model is still adequate with this abstraction.

All packets (including idle packets) carry flow control information, so one would expect that modeling of idle packets is required for proper behavioral specification. Instead, it is not required if we also abstract from the in-band structure of flow control. If we are able to model correct flow control communication behavior without the need for idle packets, the abstraction is valid. Just because flow control information is always present, only its latency is relevant and it may be modeled as a separate and direct
6.3 Modeling for Performance Analysis

In section 5.3 we already mentioned that some of the performance analysis and presentation will be performed off-line. The Markov Chain Monte Carlo method, however, needs to be performed on-line because the model may decide to stop simulation on the outcome of that analysis.
Measurement of load and throughput can be done in the input adapter process and PRIZMA, respectively. Packet delay and jitter can only be calculated when a packet has completely traveled through the system, i.e. when it has arrived at the destination process. The fact that packets carry their own time tags and possibly also fill level information, enables analysis in the destination process. In section 4.6.2 we recommended to place the analysis method in a timeless data class. Therefore, we developed a statistics-collecting data class that hides this method from the process code. This data class does not jeopardize model clarity and causes minimal pollution of POOSL code. The results of the analyses that take place in the input adapter(s), PRIZMA-T and the destination are written to an output file. In the future, these results may be presented graphically by the simulator shell around the model.

The convenient data class can be used in any process class and may analyze packets that come from any channel. A process may even possess multiple statistics-collecting data classes, for example, when analysis should be done separately on the two traffic types: guaranteed and best-effort traffic.

6.4 The Model in POOSL

This section describes the model in terms of structural entities. Figure 6.3 shows the graphical representation of the process objects in SHESIM. From left to right, we distinguish bursty sources (bsrc), input adapters (iadaps), flow control channel latency (lx2i), PRIZMA, the output adapters (oadaps) and the destination object (dest). The channels that connect the centers of all large blocks are used for the packet communication. Obviously, the channels that connect to the latency object are used to communicate flow control information. In the following sections we will briefly discuss each of these processes. A few examples of frequently used POOSL code are presented in appendix B.

Figure 6.3: Abstract system model of PRIZMA-T for performance analysis
6.4 The Model in POOSL

6.4.1 Sources

The sources are modeled by $N \times TC$ parallel threads, where $N$ is the number of switch ports and $TC$ the number of traffic classes that is supported. Behind each of the $N$ ports, $TC$ parallel threads are running independently. Note that no multiplexing takes place. POOSL allows simultaneous communication of multiple messages on the same channel. These sources were only used to test the model. Therefore, the traffic that is generated, is synthetic and uniformly distributed among the output classes.

The modeler may set the average load and average burst size for each of the ports. The load of a source ranges from 0 to 1 and this load is subdivided among the traffic classes. An example of a division of source bandwidth into four traffic classes is depicted in figure 6.4, where GU and BE denote the guaranteed and best-effort traffic types, respectively. $TC_n$ denotes traffic class $n$. The sources are specified such, that they cannot claim more than 100% of the bandwidth on the long run. That means that sources do not over-subscribe on the available link bandwidth.

A burst is an uninterrupted stream of packets that is destined for a single output port and all packets in the burst belong to one traffic class. Depending on the division of traffic class bandwidth among the two traffic types, the sources will commence a burst with packets of the guaranteed type. The second part of the burst consists of packets of the best-effort type. In order to model the bursts, we used an ON-OFF source model with a geometrically distributed ON-period and Poisson distributed OFF period. We chose for a continuous distribution for the OFF-period for reasons of simplicity and accuracy.

6.4.2 Input Adapters

The 'iadaps' process object consists of a single receiving thread and $N$ independent parallel transmission threads – one for each input adapter. Like in the real system, the queueing discipline in the input adapters is Virtual Output Queueing (VOQ) [13]. This queueing scheme has great advantages over traditional FIFO queueing. Instead of one FIFO queue, the VOQ scheme requires a queue for each of the $N$ output destinations. Incoming packets are sorted by their output destination, which solves the Head-of-the-Line (HOL) blocking problem. Note that each of the $N$ queues in an input adapter consists of a number of logical queues – one logical queue for each traffic class, similar...
Basically, we can speak of the same two-dimensional queueing structure as in PRIZMA-T. The first dimension consists of the output classes, the second consists of traffic classes. In the implementation of the switch, these queues are structured as (virtual) output queues which contain logical queues. However, the structure of the queues in the real system does not have to match with the structure of our behavioral queues. Focusing on the questions that have to be answered, it is justified that we specify a number of "traffic class queues" that are shared by the (virtual) output queues.

No matter how the queues are structured in the model, it is very useful to have information about both dimensions in the adapters as well as the switch. Therefore, we chose to construct a two-dimensional structure of FIFO queues that can be used in all processes. Figure 6.5 shows a diagram of this structure. When a packet enters the queueing structure, it is placed in the proper FIFO queue according to the output class and traffic class it belongs to. Each output class shares one queue with each traffic class,
which results in \( N \times TC \) queues. If the process places a packet in one of these queues, indexed by an output class number and a traffic class number, the administration of both dimensions is updated. According to the flow control information that is received a selection scheme may select a packet for transmission. This flow control information is based on the status of another process and may in turn be based on information of both dimensions.

The two-dimensional queueing structure is equipped with a set of data methods that allows for compact specification of the process classes that use this structure. The process may ask the data structure, for example, whether its queues contain any packets and if so, how many. It may also ask whether its threshold status has changed or whether it has a valid candidate packet that may compete in the selection scheme. These questions can be asked in both dimensions. An important aspect from a simulation perspective is that the evaluation of those questions does not depend on the size of this queueing structure. The time needed for a this data object to return the result is \( O(1) \). The credits for this achievement are in the object-oriented nature of POOSL, which has proved to be very useful.

As described in section 4.2, our modeling view focuses on keeping the real-time specification compact and clear and aims to hide timeless functional behavior in data classes. This view is used again for packet selection schemes in all processes. The selection scheme is specified in a data class and can therefore be evoked rather conveniently from the process specification. Addition of a new selection scheme does not make the real-time part of our model any more complex.

In section 7.2 a recommendation will be made on the use of this two-dimensional queueing structure for the development of flow control schemes. So in fact, the model of the architecture specification can be used as a model for future product design.

### 6.4.3 PRIZMA-T

The process class that specifies the switch is not all that different from the specification of an input adapter(!). It makes use of the same two-dimensional data structure (see figure 6.5). Only a few differences exist between the switch and an input adapter:

- The switch performs statistical multiplexing, whereas an input adapter only forwards the packets.
- The switch contains a counter on the total number of packets that is inside the process. This is the abstraction of the shared memory that we discussed previously.
- The switch uses threshold information of both dimensions of the two-dimensional queueing structure. It contains an extra parallel process for communicating any changes in the status of the switch.
- The switch drives all its outputs at the same instant; that is, there are no phase differences between the packet flows at the transmitting side of the switch.
The switch is based on the same concepts as an input adapter. It is based on the decision moment calculation approach for packet modeling. It contains a single timeless input process that satisfies receptional willingness and consists of $N$ independent parallel output processes (see figure 6.1). Finally, the switch may also use a pluggable packet selection scheme that is provided by a timeless data class, as we discussed previously.

### 6.4.4 Output Adapters

Again, the same two-dimensional queueing structure that was shown in figure 6.5 is used in the output adapters. It is built upon the concepts that we used for the specification of input adapters.

### 6.4.5 Destination

This object collects all packets in a single threaded process, without distinguishing between them. All relevant information about performance analysis in this process has already been mentioned in sections 4.6 and 6.3.

### 6.4.6 Latency Process Class

The latency process object specifies the in-band behavior of flow control communication. Its functionality is placed in a separate process class to prevent pollution of the input adapter and switch processes. The basic function of this process is delaying flow control information. For reasons of confidentiality, we cannot discuss the details of the different types of flow control that are exchanged between PRIZMA-T and the input adapter, but we may add that latency behavior can be specified independently for each type of flow control.

In the modeling phase we did not model the flow control information that is communicated between the output adapters and the switch. The reason for that is that this flow control is only important in speed-escalated configurations of the switch system. Nevertheless, all flow control modeling decisions are valid for that type of flow control and the same direct channel and latency object may be used.

### 6.5 Simulator Performance

The direct POOSL interpretation of SHESIM is not fast enough for our extensive simulation purposes. Therefore, a behavior preserving transformation to C++ is developed [14] that is based on the concept of Process Execution Trees (PET) [15]. After transformation and compilation by a C++ compiler, a speed increase of $80\times$ to $100\times$ is achieved, depending on the size of the model.

After the speed increase, a complete 32-port PRIZMA-T that supports 4 traffic classes may be simulated at a speed of roughly 650,000 packets per hour on a 450MHz Pentium II system.
6.6 Conclusions

A modeler should spend sufficient time on the details and prescribed implementation issues of a product's architecture specification. Without those details, he will not be able to evaluate the validity of his abstractions. However, it is important to resurface from the detailed level – a model that is adequate but redundant may exclude conceptual design solutions. System level modeling is about the difference between 'how' and 'what'. The modeler should be constantly aware that he models a behavior and not the implementation of this behavior.

We conclude that, in the problem context of performance analysis, the behavioral structure of the model may deviate from the actual architecture structure of the system. The best example of this is that we model the behavior of in-band flow control, but not the structure. The packet flow modeling method that we developed during our general modeling considerations enabled this abstraction to be valid.

The development of a two-dimensional queueing structure greatly benefits the specification of the model. It can be used inside the input adapters, the switch itself and the output adapters. Additionally, it opens new possibilities for the development of new flow control schemes in the PRIZMA context. This example shows how a model of an architecture can be used as a model for the design of a new architecture. This is a benefit that comes with an extensive abstraction phase.

Actual performance analysis for investigation of flow control schemes has not been performed. The abstract system model, however, is equipped to do such an analysis. Because of the lack of simulation results, we cannot decide on the adequacy of a fully asynchronous model.
Chapter 7

Conclusions and Recommendations

7.1 Conclusions

In this thesis we have shown a modeling approach that enables us to conquer the ever increasing complexity of communication systems. In addition to the high complexity of the system itself, the modeler should choose an appropriate alternative from countless modeling possibilities. The key to this puzzle is a proper formulation of the problem definition. Prior to the actual modeling phase, the modeler should relate general modeling issues to the system properties that are to be evaluated.

These general modeling issues comprise of the formulation of a modeling view, discussions about communication and concurrency, consideration of the location and method of modeling temporal behavior. In addition one may consider parametrizability and the collection and presentation of simulation results. Relating the general modeling issues to the questions at hand shapes a foundation for the modeling phase.

The modeling phase itself should commence with a round-trip into details and prescribed implementation issues of the product's architecture specification. Only careful analysis of this specification may justify the validity of abstractions that are made. If the modeler focuses on what the system is supposed to do, instead of how its behavior is implemented, proper abstractions from the details can be made.

Different groups of abstractions can be distinguished. One should start with intuitive abstractions like the specification of time and data granularity. Subsequently one may consider abstractions from architectural structure, but these abstractions should only be made if they lead to more compact, discussible or more efficient models. In this project we found that many valid abstractions could be applied to communication and concurrency, of course in the context of questions about performance properties.

Performance analysis of complex communication systems such as PRIZMA-T cannot be carried out by purely analytical methods; that is, within the development time of the product. The state space of the Markov chain that is implicitly defined by POOSL specifications explodes, even for models of modest complexity. The number of states in the Markov chain becomes so large that calculation of state equilibria (or functions thereof) is no longer possible. Therefore, one has to estimate these equilibria by applying an empirical method to one or multiple traces of the Markov chain. In this project, we applied the Markov Chain Monte Carlo method that provides an estimate of performance properties, as well as a confidence interval for the accuracy of the results. The modeler will no longer have to guess for an appropriate simulation length.
Conclusions and Recommendations

or the accuracy of simulation results. A single execution of our POOSL model of the
PRIZMA-T system yields estimations for load, throughput, delay and jitter. Methods
are under development that may also estimate memory and queue occupancy levels.
The latter can be used for the evaluation of the availability of resources for different
traffic classes.

Even with its relatively simple syntax, POOSL provides functionality that is re­
quired to evaluate the performance properties that apply to the PRIZMA system. The
concepts of time stamping and atomicy are absolutely necessary to specify a model
that is able to answer the questions. Above all, it is not difficult to learn POOSL.
We spent approximately four weeks on mastering POOSL and its underlying system
level design methodology. Another four weeks were spent on the understanding of
PRIZMA-T from its architecture specification. The remaining time was consumed by
formulating general modeling issues, matching these with the questions that were asked
and building a model after thorough abstraction from the architecture specification.

We noticed that industry tends to have a different view towards modeling than the
academic world. This project showed that neither of them are wrong. Initially, we
built a model of an architecture. The usability of this model for the design of a future
architecture greatly depends on the abstraction level that the model is specified at. Our
model may serve design decisions of future PRIZMA systems.

During the modeling process we discovered several issues that may contribute to
future research on POOSL and PRIZMA. Recommendations are made in the following
section.

7.2 Recommendations for Future Research

• We already discussed that, prior to the modeling phase, one should clearly formu­
late the questions a model should answer. It is rewarding to group those questions
in such a way that we do not have to build a new model for each question. We
attempted to group questions at a similar level. An example is the integration of
“traditional performance analysis” with questions on flow control schemes.

• In order to speed up the creation of an abstract system model, initial system
architecture specifications should contain a section on communication and con­
currency. Although the modeler should still spend time on the details of that
document, clearly describing these two areas will facilitate modeling greatly.

• It may be rewarding to use real measured traffic instead of synthesized traffic in
POOSL models. Perhaps, synthesized traffic is an abstraction that we cannot
afford for certain questions about performance analysis.

• In many cases, a product developer may wish to provide a model of the product to
its customers. If POOSL is to become a player in the field of exchanging models
for the reuse of Intellectual Property (IP), we recommend research into support of
the Open Modeling Interface (OMI) that is defined by the Open Modeling Forum
7.2 Recommendations for Future Research

(OMF). The interface is known as IEEE standard 1499. However, the difficulty here is that the quality of the verification of a combined model will suffer from the combination of informal with formal specification and description methods.

- We recommend that POOSL simulators are supplied with an analysis shell. In this way, POOSL models will not have to carry out performance analysis by themselves, but the shell will contain this functionality by passively probing into the model and necessary post-processing of simulation results.

- For reasons of confidentiality, some recommendations have been left out of this public document.
Conclusions and Recommendations
Acknowledgements

I wish to express my gratitude to the IBM Zürich Research Laboratory for providing me the opportunity to carry out this project in their advanced research environment. Special thanks go to Ronald Luijten, Ton Engbersen, professor Mario Stevens, Jeroen Voeten, Piet van der Putten and Ad Verschueren for all efforts that lead to the realization of this project. More credits go to Leo van Bokhoven and Marc Geilen for their (tool) support, Rinus van Weert for initially providing me with information on the PRIZMA system and the members of the IBM PRIZMA group for useful discussions and support. Thanks to my roommates and “neighbors” at the laboratory and university for having a good time and to Anouschka van Loon for making my stay in Switzerland a pleasant one. Many thanks to my roommates in the Monarchstraat for having a great time during my study. Finally, I wish to thank my parents and family for their love and support.

Mark Verhappen,
November 30, 1999.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCS</td>
<td>Calculus of Communicating Systems</td>
</tr>
<tr>
<td>CRC</td>
<td>Cyclic Redundancy Check</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>HOL</td>
<td>Head of Line</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol or Intellectual Property</td>
</tr>
<tr>
<td>OMF</td>
<td>Open Modeling Forum</td>
</tr>
<tr>
<td>PET</td>
<td>Process Execution Trees</td>
</tr>
<tr>
<td>POOSL</td>
<td>Parallel Object-Oriented Specification Language</td>
</tr>
<tr>
<td>PRIZMA</td>
<td>Packetized Routing in Zurich’s Modular Architecture</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SHE</td>
<td>Software/Hardware Engineering</td>
</tr>
<tr>
<td>SHESIM</td>
<td>Software/Hardware Engineering Simulator</td>
</tr>
<tr>
<td>VOQ</td>
<td>Virtual Output Queueing</td>
</tr>
</tbody>
</table>
Bibliography


Appendix A

POOSL Syntax

The syntax of POOSL process statements is given in table A.1. The syntax for data statements (DS) and data expressions (DE) are described in tables A.2 and A.3.

Table A.1: Syntax for POOSL process statements

<table>
<thead>
<tr>
<th>Process Statement PS</th>
<th>Description</th>
<th>Action</th>
<th>Time Passage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch?m(p₁,...,pₙ</td>
<td>DE) {DS}</td>
<td>message reception</td>
<td>communication</td>
</tr>
<tr>
<td>ch!m(DE₁,...,DEₙ) {DS}</td>
<td>synchronous send</td>
<td>communication</td>
<td>yes</td>
</tr>
<tr>
<td>delay(DE)</td>
<td>delay statement</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>timestamp t</td>
<td>read model time</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>while E do PS od</td>
<td>repetition</td>
<td>internal</td>
<td>no</td>
</tr>
<tr>
<td>if E then PS₁ else PS₂ fi</td>
<td>selection</td>
<td>internal</td>
<td>no</td>
</tr>
<tr>
<td>DS {DS}</td>
<td>data statement</td>
<td>internal</td>
<td>no</td>
</tr>
<tr>
<td>PS₁; PS₂</td>
<td>sequential compos.</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>sel PS₁ or ... or PSₙ les</td>
<td>choice statement</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>par PS₁ and ... and PSₙ rap</td>
<td>parallel compos.</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>PS₁ interrupt PS₂</td>
<td>interrupt statement</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>PS₁ abort PS₂</td>
<td>abort statement</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>[DE] PS</td>
<td>guarded command</td>
<td>dependent on consituents</td>
<td></td>
</tr>
<tr>
<td>m(DE₁,...,DEₙ)(p₁,...,pₙ)</td>
<td>method call</td>
<td>dependent on consituents</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Syntax for POOSL data statements

<table>
<thead>
<tr>
<th>Data Statement DS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := DE</td>
<td>assignment to variable or parameter</td>
</tr>
<tr>
<td>DS₁; DS₂</td>
<td>sequential composition</td>
</tr>
<tr>
<td>while DE do DS</td>
<td>repetition</td>
</tr>
<tr>
<td>if E then DS₁ else DS₂ fi</td>
<td>selection</td>
</tr>
<tr>
<td>DE</td>
<td>data expression</td>
</tr>
</tbody>
</table>
# Table A.3: Syntax for POOSL data expressions

<table>
<thead>
<tr>
<th>Data Expression $DE$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>data object referenced by $x$</td>
</tr>
<tr>
<td>new($C$)</td>
<td>newly created data object of class $C$</td>
</tr>
<tr>
<td>self</td>
<td>data object that evaluates this expression</td>
</tr>
<tr>
<td>$DE_1(DE_1, \ldots, DE_n)$</td>
<td>method call to method $m$</td>
</tr>
<tr>
<td>0, -1, a, 3.14, true</td>
<td>constants of primitive classes</td>
</tr>
<tr>
<td>nil</td>
<td>undefined data object</td>
</tr>
</tbody>
</table>
Appendix B

POOSL Code Examples

B.1 Parallel Threads

This section shows how to set up a number of parallel threads. As an example, the parallel output processes of the PRIZMA-T model are given. This example shows how parallel uniform behavior only needs to be specified once. First, the method startoutputprocesses has to be called, which starts as many sendpackets methods as the number of ports in the system. The sendpackets method that is shown below, is indexed by the outputportnumber and calls itself by its index in a tail-recursive manner. The second parameter (outputqueues) is also passed to the method because it does not change during the execution of the parallel threads.

\[
\text{startoutputprocesses}(\text{outputportnumber:Integer})()
\]

par

sendpackets(outputportnumber,
    outputqueues get(outputportnumber))();

and

if (outputportnumber < numofports) then
    startoutputprocesses(outputportnumber+1)()
fi;

rap.

sendpackets(outputportnumber:Integer;
    outputqueues:LogicalQueues)()

... sendpackets(outputportnumber, outputqueues)().
B.2 Guards

This example shows how the sendpackets method can be triggered to start execution in the event of a non-empty queue (see also figure 6.1 on page 40). For this, a guard is used. The guard process statements are one of the most powerful statements in POOSL.

\[
\text{sendpackets}(\text{outputportnumber}: \text{Integer}; \text{outputqueues}: \text{LogicalQueues})() \\
\quad [\text{outputqueues are not empty}] \\
\quad \ldots \\
\quad \text{pout!packet}(\text{packet}) \{ \ldots \}; \\
\quad \text{sendpackets}(\text{outputportnumber}, \text{outputqueues})().
\]

B.3 Atomicity

In the POOSL code below, we show how atomic statements are used in combination with a communication action. When a packet is transmitted, the switch's administration should be updated immediately. To prevent other urgent actions from accessing this data before it is updated, the atomic statement can be used. The example shows that the data statements between curly brackets are executed immediately upon communication of the packet. A receiving process may use atomicity in a similar way.

\[
\text{sendpackets}(\text{outputportnumber}: \text{Integer}; \text{outputqueue}: \text{LogicalQueues})() \\
\quad \ldots \\
\quad \text{pout!packet}(\text{packet}) \{ \\
\quad \quad \text{outputqueue remove}(\text{packet getpriority}); \\
\quad \quad \text{sharedmemory packet minus} \\
\quad \}; \\
\quad \text{sendpackets}(\text{outputportnumber}, \text{outputqueues})().
\]
B.4 Pluggable Mechanisms

The example below shows a “pluggable” packet selection scheme that is specified in a timeless data class. Different selection mechanisms can be given as parameters to the data class. Just before transmission of the packet in the example, the method select of our LogicalQueues data class is called, which returns that particular packet.

```plaintext
sendpackets(outputportnumber:Integer;
outputqueues:LogicalQueues)()

...  
packet := outputqueues
   select(selectionmechanism,time asInteger);
   pout!packet(packet) { ... };
   sendpackets(outputportnumber,outputqueue)().
```

B.5 Decision Moment Calculation

As discussed in chapter 4, the decision moment calculation principle can be specified elegantly in POOSL. The example below shows how the calculation is started in the event of a non-empty queue. The timeuntildecision is given back to the sendpackets method, which delays for that amount of time. Because of the specification of the calculatedecisionmoment method, it is made sure that packets are transmitted at periodic instants in time.

```plaintext
sendpackets(outputportnumber:Integer;
outputqueues:LogicalQueues)()

[ outputqueues arenotempty ]
   calculatedecisionmoment(){timeuntildecision};
   delay(timeuntildecision);
   ...
   pout!packet(packet) { ... };
   sendpackets(outputportnumber,outputqueue)().
```
calculatedecisionmoment() (timeuntildecision:Integer)

timestamp time;
integertime := time asInteger;
timeuntildecision := packetcycle;

if (integertime % packetcycle != 0) then
  time_decision := (packetcycle*((integertime/packetcycle)
                    floor + 1));
  timeuntildecision := timeuntildecision +
                        time_decision - integertime;
fi.