2nd International Workshop on Advanced Teleservices and High-Speed Communication Architectures,
26.-28.09 1994, Heidelberg, Germany

CPU Utilization of Multimedia Processes: HeiPOET – The Heidelberg Predictor of Execution Times Measurement Tool

Hartmut Wittig
Lars C. Wolf
Carsten Vogt

IBM European Networking Center, Vangerowstraße 18, D-69115 Heidelberg
Mail: { wittig, lwolf }@vnet.ibm.com, vogt@fh-koeln.de

Abstract: Due to the time characteristics of audio and video data, the processing of multimedia applications has to be done using real-time mechanisms. Scheduling algorithms used within such systems require information about the processing time requirements, the CPU utilization of the applications, to perform schedulability tests. Since multimedia applications are often constructed by combining processing modules (often called stream handlers), processing time determination for these modules is required. The multitude of these modules and the large variety of computer systems calls for a measurement tool. In this paper we define the term CPU utilization for multimedia processing and describe the CPU utilization measurement tool HeiPOET. The presented measurements show that the tool provides measurement results with good accuracy.

1 Introduction

Multimedia applications have characteristic quality-of-service (QoS) requirements that must be satisfied by the underlying computer system, i.e., its local and its network resources. The CPU, being the principal local resource to be managed, requires functions for workload policing, capacity reservation, and process scheduling to enable the processing of real-time multimedia data. In general, application generated CPU workloads can be distinguished into the following service classes:

- non real-time processes
- control processes handling real-time and non-real-time tasks
- real-time processes with soft deadlines
- real-time processes with hard deadlines

The first class contains processes without deadlines, for example a compiler process. Because these applications have no real-time requirements they may run with the lowest priority among these classes. In most CPU reservation and scheduling schemes, at least a small amount of CPU capacity will be reserved for such applications to avoid starvation. This part of CPU capacity can be managed using classical strategies such as round robin or multi-level feedback.

Control processes regulate the access to certain resources. For example, an important control process for the CPU is the scheduler. A control process can also be used to reserve network resources in the setup phase of multimedia communication. On the other hand the same control process can be involved in the transmission of real-time messages, e.g., acknowledgment messages to trigger retransmissions of lost or corrupted data units [6]. The control processes should be handled in a deterministic manner with high priority.

Real-time applications with hard deadlines (e.g., manufacturing control systems) are processed in dedicated systems. For these systems exists a great variety of scheduling and reservation mechanisms ([5]). Though it is difficult to integrate hard real-time systems with multimedia applications, it can be done in principle ([12]).

Most multimedia processes can be considered as soft-real-time processes. If a data unit is too late or lost, this is not necessarily noticed by the human viewer. To avoid disruptions a characteristic quality of the media presentation is required. To describe these processing requirements of multimedia applications, a characteristic QoS parameter set must be defined. For example, the playback of a video stream requires a picture loss rate below a certain percentage, and it is necessary that each logical data unit (e.g., video frame) is displayed at a certain point in time. In a multimedia system resource reservation mechanisms and QoS enforcement strategies are used to guarantee QoS requirements of multimedia applications. This paper focuses on multimedia applications which process continuous media with soft deadlines.

CPU scheduling of multimedia processes is based on the classical real-time scheduling algorithms, Rate Monotonic (RM) [11] and Earliest Deadline First (EDF) [11]. RM scheduling is a static scheme which is defined in a context of tasks that require periodic CPU processing. Therefore, RM scheduling is well-suited for applications processing continuous media. According to their processing periods, processes have different, fixed priorities. Processes with small periods are executed with the highest priority. Low priorities are assigned to processes with large periods. At any time, the dispatcher of the CPU chooses the highest-priority process to run next. EDF scheduling is a dynamic real-time scheduling algorithm. The task with the earliest deadline has the highest priority. Because the timing constraints of all processes change, the priorities of tasks are dynamically adapted. To test whether a new task can be scheduled using RM or EDF scheduling, the following condition must be met:

$$\sum_{\forall i \in Tasks} R_i \times P_i \leq B$$

Index i runs through set of all multimedia tasks, R_i denotes the maximum processing rate of task i, P_i is the processing time per-period, and B is the schedulability bound. For RM scheduling the schedulability bound is determined by $B = \ln(2) \approx 0.69$. The schedulability bound of EDF scheduling is $B = 1$.

Each multimedia application has to specify the workload it will generate. This workload specification consists of the processing rate R and execution time per period P. The resource management system performs the schedulability test to decide whether this new application can be accepted. If enough CPU capacity is available to execute the new application without disturbing existing applications, the schedulability test returns successfully. As seen from the above formula, the processing time is an important parameter for the schedulability test. However, the determination of these values is still a largely unresolved issue in the area of CPU scheduling. The main problems measuring CPU utilization of multimedia applications are:

- Great variety of multimedia applications and interfaces
- Exact definition of the term "CPU utilization" of multimedia applications
- Handling of Variable-Bit-Rate Streams

This paper focuses on the problem of how to determine CPU utilization of multimedia applications. The HeiPOET (Heidelberg Predictor of Execution Times) measurement tool provides a computer-based prediction of CPU utilization and a refinement of measured values during the execution time of multimedia applications. The highlights of the HeiPOET tool are:

- Automation of CPU utilization measurements
- Specification language for the description of various parameters
- High precision of measurement results by eliminating operating system interruptions
- Computation of statistical metrics to check the reliability of measurements
- Re-usability of measurement results by storing measurements in a database

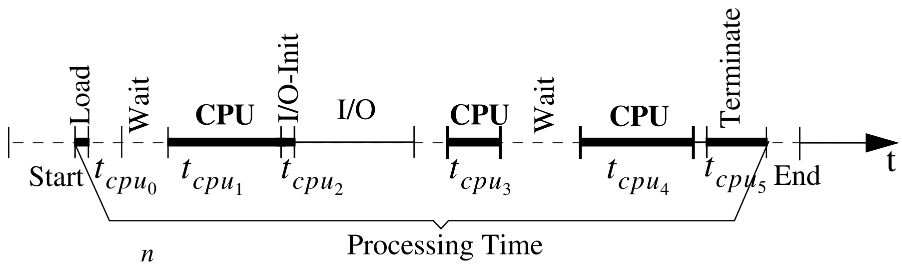
In the next section, we describe the model chosen for CPU utilization. Then design principles of HeiPOET are explained. Measurements presented in Section 4 show the validity of the architecture before we give our conclusions.

2 CPU Utilization of Multimedia Processes

In this section a definition of the term "CPU utilization" of multimedia processes and a characterization of stream handlers as the heart of multimedia applications are given.

We define the CPU utilization of an application process as follows:

The CPU utilization of an application process is the overall duration in which the CPU is occupied in order to perform this application task.



$$t_{cpu} = \sum_{i=0}^{n} t_{cpu_i}$$

t_{cpu_i}...Duration of i-th processing cycle
t_{cpu}... Overall CPU utilization of the multimedia module

Fig. 1. Definition of CPU utilization

In Figure 1 it can be seen that CPU utilization consists of two different parts. In the first place, it includes the pure application code execution time on the CPU. Additionally, there are operating system activities to make the execution of multimedia applications possible, e.g., context switches, initialization and termination of I/O operations. An example is a multimedia application which plays an MPEG video stream coming from the hard disk. Most parts of the CPU are utilized for the software decompression algorithm, yet the read operation also needs CPU time, at least to set up an asynchronous disk I/O operation to read the multimedia data.

The operating system activities are characterized by the multimedia application process model (see Figure 2) derived from the classical UNIX process model ([2]). To determine the CPU utilization of a periodic multimedia application, three states are of specific interest: the states “Running”, “Asleep” and “Ready to Run”. If a process is in the state “Running” its machine code will be executed. The time an application is in this state is the prominent part of its CPU utilization. The transitions between these states represent the operating system activities which also require CPU capacity. Thus, initialization, termination, context switch, wakeup and sleep operations (①-⑥) are the operating system parts in the definition of the CPU utilization of a multimedia application. In the life cycle of process initialization (①) and termination (⑥) operations are system activities which are performed only one time, namely at the beginning and the end of the multimedia processing. Because the initialization and the termination of a multimedia application are not time-critical, and periodic scheduling algorithms do not consider aperiodic parts in the schedulability test. In the context of this paper, these operations are of no interest for the measurement of CPU utilization.

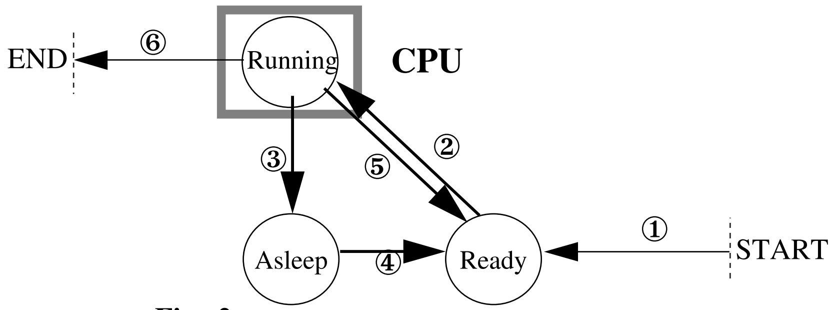


Fig. 2. State transition diagram of multimedia processes

The CPU utilization of an application is not fixed. The following formulae gives an abstract description of factors which influence the CPU utilization:

$$t_{cpu} = f(Sys,\ In,\ Dyn)$$

CPU utilization depends on the type and performance of the CPU in a computer system (parameter *Sys*). The greater the performance the lower is the CPU utilization of a specific multimedia application. It also depends on the kind of data flowing into the multimedia application, e.g., control information like the picture rate of an MPEG stream, or the compressed data stream containing the MPEG video (parameter *In*). A third influence to the CPU utilization are dynamic properties of the operating system, e.g., the scheduling algorithm and timing (parameter *Dyn*). For example, using preemptive scheduling algorithms causes more context switches than non-preemptive scheduling algorithms.

Often multimedia systems are structured in modules, objects or devices (e.g.,

[9],[3],[1],[13]). Each module is responsible for a logical operation on the continuous-media stream, e.g., read operation from a multimedia file system, mixing of audio and video streams. These modules are called stream handlers (SH) (e.g., [9],[13]). In general, an SH is a software entity processing continuous-media data streams. The general structure of an SH is shown in Figure 3.

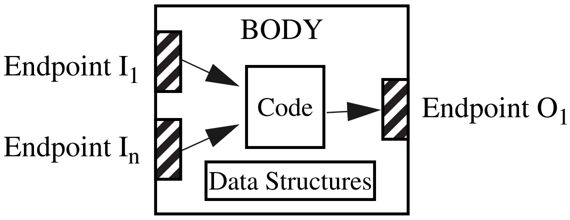


Fig. 3. Structure of SHs

An SH consists of a body and at least one endpoint. The SH body includes data structures for control information and the executable code. SHs transfer multimedia data via endpoints[1]. Data flows unidirectional between SHs. Endpoints for sending data are called output endpoints, endpoints for receiving data are input endpoints.

A multimedia application can be modelled as an acyclic directed graph of SHs beginning with source SHs and ending with sink SHs. The SH graph is constructed at the application start-up time and can also be changed dynamically at application run time.

The CPU utilization of a multimedia application consisting of SHs is the sum of the CPU utilization of each SH in the SH graph. Hence, for the CPU utilization measurement the application can be divided into its basic parts, the SHs, which are then measured individually. The advantage of this approach is its modularity, which reduces the measurement effort. SHs are well structured and easy to handle for measurements. Measuring the CPU utilization based on SHs allows to handle dynamic changes in the multimedia application. E.g., if a new SH is added to the existing graph at application run time, the CPU utilization of the new SH can obviously be added to the CPU utilization of the SH graph.

3 Design Principles and Architecture

HeiPOET is a measurement tool for CPU utilization of multimedia SHs. The architecture of HeiPOET is based on related work in the field of measuring response times in hard real-time systems (see [7, 8, 10]), however, since response time is not of primary interest for multimedia applications, the results of this work from hard real-time systems is not usable in multimedia systems. The design of the HeiPOET measurement tool is shown in Figure 4.

1. An exception are source and sink SHs: the source SH receives data from device drivers and the sink SH sends data to device drivers.

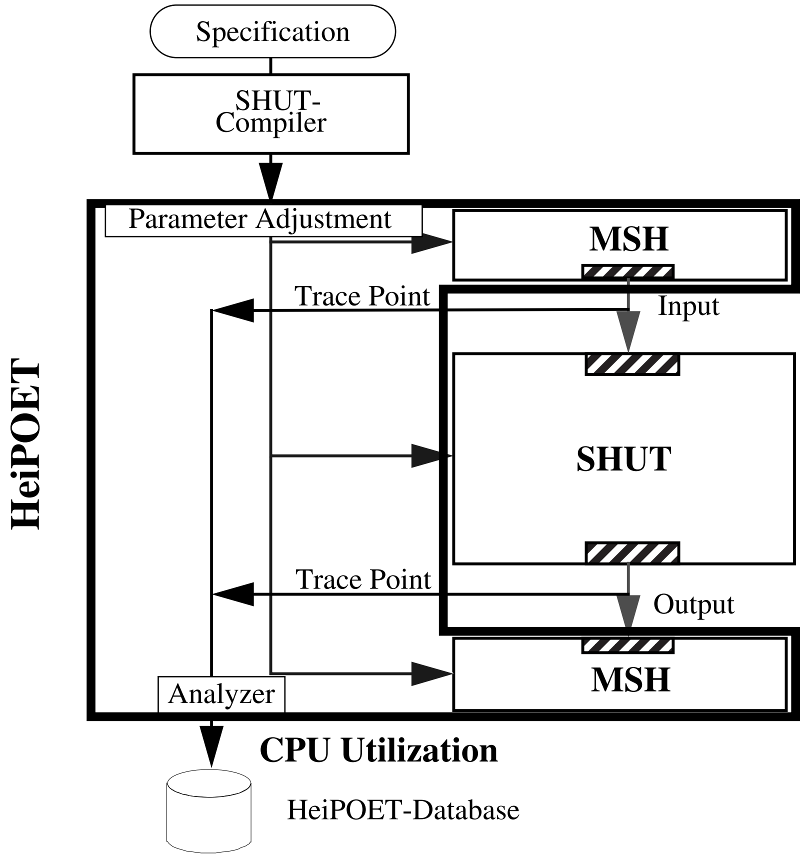


Fig. 4. Design of the HeiPOET measurement tool

HeiPOET is structured into three major parts:

- SH specification language and compiler: A language describes a set of SH initialization values to measure the SH under various input parameters and conditions. All control attributes of SHs can be modified using this specification language. Thereby, the SHs can be adjusted on the fly and CPU utilization can be measured using various parameter settings of the SH. A description of the SH specification language is given in Appendix A.
- Measurement stream handlers: Specific SHs to send/receive test data to/from the stream handler under test (SHUT) are called measurement stream handlers (MSH). They are used to form the environment in which the SHUT is embedded in the measurement phase. Because of the variety of SHUT the MSHs must be flexible. According to the elements in the specification language the SHUTs, the input MSH and the output MSH are adjusted. Major parameters to be adapted are the number of input and output endpoints of the SHUT, the number of incoming and outgoing packets of each endpoint per processing period, and the size of incoming packets.
- Analyzer: The analyzer reads the recorded measurements, analyzes the measure-

ments, computes statistical metrics, and stores the results in the HeiPOET database.

The HeiPOET tool works as follows: HeiPOET reads a SH specification file containing the description of a set of SH initialization values (see Appendix B). The MSHs and the SHUT are initialized with the first set of specified initialization values. The sequence of operations in the HeiPOET measurement phase is shown in Figure 5.

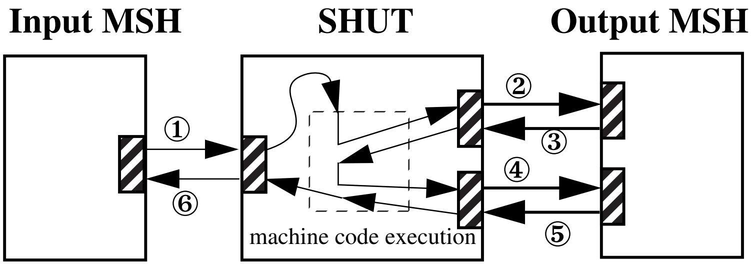


Fig. 5. HeiPOET measurement phase

Depending on the actual scenario, the input MSH sends a data frame to the SHUT. When the control flow changes from the MSH to the SHUT, a timestamp is recorded using a system trace mechanism (①), and the machine code of the SHUT is executed. When the SHUT finishes the processing for the current data packet (i.e., the processed packet is sent to the other MSH), a second timestamp is saved (②). The output MSH receives the data and the control flow immediately returns to the SHUT. This is the beginning of the next SHUT activation, a new timestamp is recorded (③). After sending data packets through other endpoints to the output MSH and saving the timestamps of these operations (④,⑤)the control flow returns from the SHUT to the input MSH (⑥). The duration between the timestamps ① and ⑥ is a complete processing period of the SHUT. The SHUT is in the state “Running” between timestamps ① - ②, ③ - ④, and ⑤ - ⑥. This time can directly be added to the CPU utilization of the SHUT.

Interrupts and other system operations (e.g. sleeping, paging, swapping) in the execution phase of the SHUT are also recorded by the trace facility (the whole kernel of the used AIX™ operating system is instrumented with calls to trace functions). Thereby, it is possible to detect, to measure, and to subtract all interruptions occurring during the measurement phase of the SHUT (e.g., by higher priority processes or interrupt handling routines). This technique guarantees measurement results with high accuracy. Analyzing the resulting trace with the saved timestamps, HeiPOET can compute the CPU utilization of the SHUT. To increase the precision of the measurement, the measurements are repeated for each set of SHUT parameters for a certain number of times. Statistical metrics, such as mean CPU utilization, standard deviation, and the 0.95 confidence interval, are computed to get information about the reliability of the CPU utilization measurements.

These steps are repeated until the CPU utilization for each set of parameters in the SH specification file has been measured. All results are saved in the HeiPOET database.

Since the CPU utilization of a SH depends on the performance of a computer, it is necessary to measure the CPU utilization on each computer system. E.g., the measurements can be done at SH installation time and when the system configuration changes. As part of the workload specification of an applications SH graph, the CPU utilization values of the SHs are retrieved from the HeiPOET database.

4 Measurement Results

Using an implementation based on the described architecture, measurements have been performed to validate the HeiPOET approach.

The first measurement object is a multicast SH. Multicast SHs can be used in multimedia conferencing systems to copy one incoming stream to many partners. A multicast SH has one input endpoint and many output endpoints. The number of output endpoints can dynamically be changed, e.g., if partners join or leave a multimedia conference. The first measurement is based on a multicast SH with two output endpoints. In one processing period, the multicast SH duplicates a packet (10 KByte) coming from the input MSH and forwards them through two output endpoints to the output MSH. The CPU utilization measured with the HeiPOET tool[2] is shown in Figure 6.

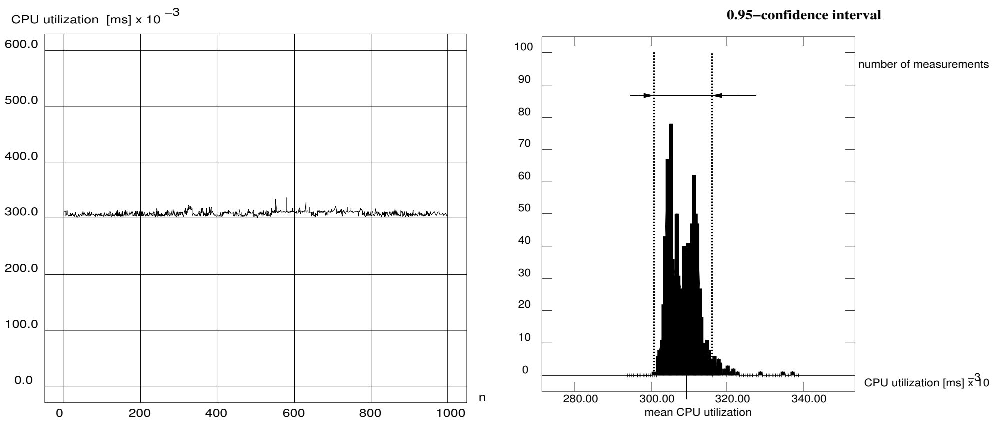


Fig. 6. CPU utilization of a multicast SH with two endpoints

The diagram on the left side shows the measurement series. Though the test machine was heavily loaded, it can be seen that the CPU utilization is approximately constant. The diagram on the right side of Figure 6 gives statistical metrics to quantify the confidence of the measured results. The mean CPU utilization of the multicast SH is 310µs. The standard deviation in the measurement series is 3µs, this means less than one percent of the mean CPU utilization. The 0.95 confidence interval is [301µs, 316µs].

Additional measurements on various computer models are presented in Table 1. These measurements are based on a mixer SH which mixes two incoming streams and produces one outgoing stream (e.g., to mix digital audio streams coming from microphones). It can be seen from Table 1 that the standard deviation is very small.

---

2. The measurements were performed on an IBM RISC System/6000 Model 530™ running AIX 3; the system was used by other tasks and users during the measurement.

In all cases the standard deviation has never exceeded three percent of the mean CPU utilization. It can be resumed that the HeiPOET tool works with a very high precision. In practice, HeiPOET is sufficient to measure the CPU utilization of SHs.

Tabelle 1: CPU utilization of a mixer SH

Computer (RISC System/6000)	mean CPU utilization [ms]	standard deviation [ms]
Model 360	8.0	0.02
Model 340	12.2	0.04
Model 530	16.1	0.03

Finally, a multicast SH has been measured with a varying number of output endpoints. Using the SH specification language measurements are performed with a number from two to ten output endpoints (see Appendix B for the SH specification file). Figure 7 shows the results. Each point in the diagram represents 1000 single measurements. It can be seen that there is a linear relation between the number of output endpoints and the CPU utilization of the multicast SH. Because of the complexity *O(n)* of the multicast operation (where *n* is the number of output endpoints), these measurements can be taken as an additional proof for the correct work of the HeiPOET tool.

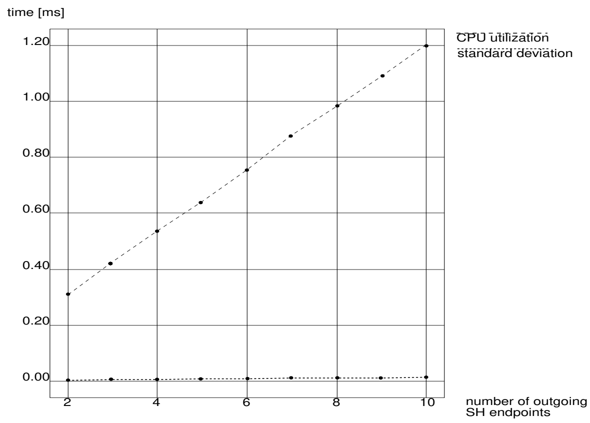


Fig. 7. Multicast SH with two to ten endpoints

5 Conclusions

HeiPOET was developed to solve the problem of CPU utilization measurement of multimedia applications. Based on the CPU scheduling algorithms and the SH model, the architecture of HeiPOET was designed. Using the SH specification language, HeiPOET is suitable to measure SHs with various numbers of inputand output endpoints. The modular SH model is used to split a multimedia application into single stream handlers, measure CPU utilization of each SH, and combine these measurement results to the overall CPU utilization of the multimedia application. The measurements have shown that the HeiPOET tool is able to produce results with sufficient precision; mean CPU utilization and other statistical metrics show the trustworthiness of the measurements. These values are stored in the HeiPOET database, and can be used by CPU resource managers to test the schedulability of the multimedia tasks.

Further measurement will focus on SHs processing variable bit rate streams, like MPEG encoding and decoding SHs.

References

[1] M. Altenhofen, J. Dittrich, R. Hammerschmidt, T. Käppner, C. Kruschel, A. Kückes, T. Steinig: *The BERKOM Multimedia Collaboration Service*. Proceedings of the ACM Multimedia '93 Conference, Anaheim, 1993.

[2] M.J. Bach: *The Design of the UNIX Operating System*. Englewood Cliffs N.J., Prentice Hall, 1986.

[3] G. Blair, G. Coulson, P. Auzimour, L. Hazard, F. Horn, J.B. Stefani: *An Integrated Platform and Computational Model for Open Distributed Multimedia Applications*. Proceedings 3rd International Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, 1992.

[4] N. Chomsky, *Syntactic Structures.* Den Haag, Mouton, 1957.

[5] C.S. Cheng, J.A. Stankovic, K. Ramamritham, "Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey", *IEEE Tutorial on Hard Real-Time Systems*, Washington D.C., Computer Society Press of the IEEE, S. 150-174, 1988.

[6] L. Delgrossi, C. Halstrick, R.G. Herrtwich, F. Hoffmann, J. Sandvoss, B. Twachtmann: *Reliability Issues in Multimedia Transport*. Proceedings of the Second Workshop on High Performance Communication Subsystems (HPCS '93), Williamsburg, Sept. 1993.

[7] D. Ferrari, G. Serazzi, A. Zeigner: *Measurement and Tuning of Computer Systems*. Englewood Cliffs N.J., Prentice Hall, 1983.

[8] P. Gopinath, T.Bihari, R. Gupta: *Compiler Support for Object Oriented Real-Time Software*. IEEE Software, Vol. 9, No. 5, September 1992.

[9] R.G. Herrtwich: *Timed Data Streams in Continuous Media Systems*. TR-90-026, International Computer Science Institute Berkeley, May 1990.

[10] K.B. Kenny, K.J. Lin: *Measuring and Analyzing Real-Time Performance*. IEEE Software, Vol. 8, No. 5, September 1991.

[11] C.L. Liu, J.W. Layland: *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*. Journal of ACM, Vol. 20, No. 1, January 1973.

[12] S. Shenker, D.D. Clark, L.Zhang: *A Scheduling Service Model and a Scheduling Architecture for an Integrated Services Packet Network.* Internet Draft, 1993.

[13] L. Wolf, R.G. Herrtwich: *The System Architecture of the Heidelberg Transport System*. ACM Operating Systems Review, Vol.28, No. 2, April 1994.

Appendix A: Syntax of the SHUT Specification Language

The syntax of the SHUT specification language can be described with a context-free grammar G consisting of set of terminal symbols T, non-terminal symbols N, transformation rules R and a starting symbol s [4]:

$$G = (T, N, R, s)$$

The SHUT specification language follows this grammar:

T = { IN , OUT , ATTRIBUTES , ENDPOINT, LOOP , ; , , , = , TO , STEP
SUB , 0..9 , a-z , A-Z }

N = { SHUT_specification,
identification, input_descriptor, output_descriptor, attribute_descriptor,
SHUT_name, SHUT_type, endpoints, endpoinds_ea,
attribute_list, simple_list, attributes, attributs, attribute_type,
simple_type, enumeration_type, loop_type, comment }

s = SHUT_specification

The rules R of the SHUT specification language are described in Backus-Naur form:

```
<SHUT_specification> ::=
        <identification> <input_descriptor> <output_descriptor> <attribute_descriptor>
<identification> ::=
        IDENT   <SHUT_name> <SHUT_type>
<input_descriptor> ::=
        IN   <endpoints>
<output_descriptor> ::=
        OUT  <endpoints>
<attribute_descriptor> ::=
        ATTRIBUTES   <attribute_list>
<SHUT_name> ::=
        NAME  = <string> ;
<SHUT_type> ::=
        TYP = ( SYNC | ASYNC <integer> ) ;
<endpoints> ::=
        ε | <endpoint> <endpoint_ea>;
<endpoint_ea> ::=
        ENDPOINT <integer> = <integer>;
<attribute_list> ::=
        <simple_list> | <attribute_list> <simple_list>)
<simple_list> ::=
        ε | LOOP {   <attributes>  }
<attributes> ::=
        attribute | attributes attribute
<attribute> ::=
        <integer> SUB <integer>  = <attribute_type> ;
<attribute_type> ::=
        <simple_type> | <enumeration_type> | <loop_type>
<simple_type> ::=
        <integer> | <string>
<enumeration_type> ::=
        <simple_type> | <enumeration_type> , <simple_type>
<loop_type> ::=
        <integer> TO <integer> STEP <integer>
```

Comments are allowed to be inserted everywhere into the SHUT specification .

```
<comment> ::=
        ( # <string> \n ) | ( # <string> #)
```

Appendix B:
SHUT Specification of a Multicast Stream Handler

An example of a SHUT specification file of a multicast stream handler (MSH) is presented.

```
IN
     ENDPOINT 1 = 1 # one incoming packet per processing period
                      for input endpoint 1 #
OUT
     ENDPOINT 1 = 1 # one outgoing packet per processing period
                      for output endpoint 1 #
     ENDPOINT 2 = 1 # one outgoing packet per processing period
                      for output endpoint 2 #
     ...
     ENDPOINT 10 = 1# one outgoing packet per processing period
                      for output endpoint 10 #
ATTRIBUTES
 LOOP
     1002 SUB 1 = 2 TO 10 STEP 1
                      # attribute 1002 of the MSH specifies the number
                            of MSH output endpoints (2 .. 10) #
     1024 SUB 0 = 10
                      # attribute 1024 of the MSH specifies the size
                            of incoming packets #
...
```

In the MSH specification there are three different descriptors: data input, data output, and attribute descriptor.

The input characteristics of the MSH are described in the input descriptor. The input descriptor contains the number of incoming packets in one processing period of the MSH.

The output descriptor specifies the number of outgoing packets in one processing period of the MSH. According to the multicast tree of the multimedia application, the number of output endpoints varies. It is specified that there is one outgoing packet per output endpoint in one processing period of the MSH.

One of the main goals of the HeiPOET tool is the automation of the CPU utilization measurement of stream handlers with varying parameters. In the attribute descriptor various parameter settings of the MSH can be described using the loop type (see Appendix A). The loop type in the example specifies that the CPU utilization must be measured for an MSH with 2 to 10 output endpoints. The size of the incoming data packets is 10 KByte.