IS&T/SPIE Proceedings Vol.2417
Multimedia Computing and Networking, San Jose (CA), USA, February 6-8, 1995

Activation Set: An abstraction for accessing periodic data streams

Tobias Helbig, Soufiane Noureddine, Kurt Rothermel

University of Stuttgart / IPVR, Breitwiesenstr. 20-22
D-70565 Stuttgart - Germany

ABSTRACT

In distributed multimedia applications time-dependent data streams are conveyed and processed under real-time conditions. The timely accurate activation of the stream handlers processing the data units of streams requires deriving their scheduling times from the temporal properties of the data streams and the amount of data that is processed in each activation.
In this paper, we propose the *activation set* concept as a data access abstraction to consume sets of data units from a group of time-dependent data streams having synchronization relationships among each other. The concept supports the consumption of multiple periodic data streams of different rates as well as the integration of aperiodic streams. It is shown how scheduling times of activations are derived from the streams' temporal properties and the requested amount of data. When a stream handler is activated, data units of the streams are selected for consumption according to synchronization relationships and tolerated skew bounds. The effects of the approach on delays are discussed in the context of a rate-monotonic scheduling algorithm.
The concept provides a configurable interface to groups of time-dependent data streams which, for example, is required when mixing streams. It is widely applicable in multimedia system architectures that allow for configuration of distributed multimedia applications based on interconnected stream handling components.

Keywords: multimedia, time-dependent data streams, synchronization, data access, activation set, scheduling

1. INTRODUCTION

Distributed multimedia applications may be decomposed into fairly independent stream handlers, so-called components, among which the data units of streams are conveyed. Components handle the generation of data units, e.g. by capturing frames from a camera, the transformation of data units by, say, compression, decompression, or mixing, and the presentation to a user. The establishment of a multimedia application requires, on the one hand, the programming of the components' stream handling functions. On the other hand, components are interconnected to arbitrary processing topologies. These separate activities are referred to as "programming-in-the-small" and "programming-in-the-large", respectively [DeKr75].

A particular feature of multimedia applications is the transmission and processing of time-dependent data streams. Similar to the decomposition of complex application scenarios into interconnected components, the temporal control of complex processing topologies can be decomposed into timed activations of components. The temporal behavior of application topologies is controlled by application programmers. They specify and manipulate the temporal properties of data streams (e.g. their rates, starting times, or the direction of play-back) and the temporal relationships among data streams (e.g. synchronization relationships and tolerated skew). However, the actual timing of the components' activations to process the streams' individual data units should be hidden from application programmers behind the interface of a multimedia operating system. To achieve this goal, the temporal properties of each stream handling task have to be derived from the temporal specifications provided by an application programmer. Additionally, an appropriate mechanism is required to activate processing tasks on time.

This paper focuses on the basic mechanism and the system architecture which allow for the timely activation of periodic stream handling tasks and the selection of sets of data units, the so-called *activation sets*, that are processed in each activation. Components are activated in a multi-threaded environment. A component may, in general, process one or more data streams, e.g. by mixing them. Data streams may have different rates. More or less stringent synchronization relationships may apply among them. The proposed mechanism derives the scheduling times of activations from the temporal properties of the data streams that are processed and the amount of data that is requested by a component. Scheduling times are passed to a modified rate-monotonic scheduler which controls the execution of concurrent threads. Upon activation of a component, the data units of an activation set are selected depending on the chosen data selection mode. The selection modes support the processing of

data units under consideration of synchronization relationships ranging from tightly coupled streams allowing no skew at all to loosely coupled or temporally non-related streams. Selection modes may be set individually for each stream.

The approach supports the separation of stream processing functions from timing considerations of the application scenario a component is used in. In particular, components do not have to be concerned with the determination of processing times or the observation of synchronization relationships among the streams they process. A component only requests the amount of data it intends to process in its next activation. It is activated and given the requested data units at the appropriate point in time. Besides simplifying the programming of components, processing may be scaled transparently, i.e. the speed or direction of the processing of data units may be changed without giving the components notice of it.

The proposed mechanism provides a base for arbitrary flow control, synchronization or buffer control protocols. It is a non-distributed algorithm that does not enforce any restrictions regarding the origins of the data streams a component processes. Streams may originate on the same or any other node of a distributed system. The solutions proposed in this paper lay stress on the integration of transport and operating system issues. The mechanism bridges the gap between releasing data units at the edge of a transport system and their actual processing scheduled under control of the operating system.

The paper is structured as follows. In the next section, we will give an overview of related work. In Section 3, the system architecture for the activation of components is introduced. Section 4 focuses on the description of activation sets and the selection modes to decide whether or not a data unit is part of an activation set. In Section 5, we discuss the scheduling of the processing of activation sets in the context of a rate monotonic scheduling algorithm. The paper is closed with some concluding remarks.

2. RELATED WORK

Multimedia applications consist of a number of fairly independent tasks that generate, transform, or play out data units of streams. Thus, the idea of encapsulating these tasks into individual software entities and configuring applications by interconnecting the data access points of these entities is widely accepted. The concept may be found in the Conic [KrMa85] and REX [MKSD90] systems which support the establishment of general distributed applications, as well as in environments to configure multimedia applications, such as the work done by Gibbs [Gibb91], the *CINEMA* system [BDH+93], [RBH94], the multimedia system services of the IMA [Hewl93], or the SUMO project [CBRS93].

In such architectures, the problem arises how to activate software entities at appropriate times. An approach is to model software entities as active objects that take care for their own timing [Gibb91], [DNNR92]. That complicates the code of stream handlers, as they have to take care of their temporal behavior and of synchronization relationships among each other. We aim at separating the stream handling functions from the handling of the temporal behavior. In our opinion, the multimedia system environment should realize specifications of temporal behavior and synchronization relationships, allowing an application programmer to focus on the stream handling code alone. QuickTime [Appl91] separates the description of temporal properties and the execution of stream handling code by providing the feature of timed execution of callback functions. However, callback functions can only be executed properly if an application provides sufficient computing time, i.e. resource management is up to the application instead of the system environment. Furthermore, no particular timing support for mixing of arbitrary streams is offered.

In the case of periodic processing tasks, resource management of the CPU is supported by well-established scheduling algorithms, like *rate-monotonic* [LiLa73] or *earliest-deadline-first* scheduling. They allow for resource reservation by integrating schedulability tests and offer guaranteed limits on execution times. We therefore base our approach on a modified *rate-monotonic* scheduler [Bart94].

In multimedia applications time-dependent data streams are conveyed. Consequently, an appropriate modeling of the temporal properties of data streams is required as well as the realization of synchronization relationships among data streams. Besides models reflecting more complex structures of time-dependent data streams [GBT94], a simple and widely applicable approach is to model streams as a sequence of timestamped data units which are attached to a time axis and which have a certain duration [Herr91]. The preservation of temporal relationships among data streams is discussed in the context of distributed syn-

chronization protocols (e.g. in [EDP92], [RRVK92], [RoHe94]), in several mechanisms at the transport system level and in the context of media mixing in teleconferencing scenarios. To access groups of time-dependent data streams, mechanisms to synchronize related transport system connections are of particular interest. The approaches range from interleaving of related data units [NaSm92] to the introduction of synchronization markers [ShSa90]. In more recent publications, approaches are described that provide for segmentation of the data streams of related transport connections, the description of reactions on late or lost data units and protocols for delay compensation ([RaBa93], [SZS94], [BaLe94], [LKG92]). In general, the mechanisms are transport system oriented. They consider the timing of data delivery at the edge of the transport system without a discussion of interactions with the operating system's scheduling algorithm. Only in [BaLe94] an additional primitive is proposed to support a finer timing of stream handlers on top of a UNIX scheduler. Furthermore, general distributed multimedia applications may include data paths traversing multiple transport system hops requiring synchronization mechanisms above the transport system layer. Our approach tries to stress the integration aspects between transport and operating system functions in particular when mixing data streams. It provides a basic algorithm that can be used even when multiple transport system hops are traversed.

Media mixing of data streams originating from different nodes is discussed in the context of a teleconferencing scenario in [RVR93]. The term *fusion set* is introduced to refer to a set of packets being mixed in each activation step of the mixer. The paper focuses on restoring synchronization relationships in the absence of globally synchronized clocks and is based on a simplified data stream and processing model: each stream has the same rate and exactly one data unit is processed per stream and activation. The activation sets, introduced in our paper, may be seen as a generalization of fusion sets.

3. SYSTEM ARCHITECTURE FOR MODULAR MULTIMEDIA APPLICATIONS

This section describes the system architecture that allows for the periodic execution of modular stream handling tasks. We first summarize our application model (for more details see [RBH94]). Then, the layers of the system architecture to activate stream handlers are discussed in detail.

A multimedia application consists of several stream handling tasks. Each task is performed by a so-called **component**. Components process data units of multimedia data streams, e.g. they generate video frames, overlay or mix frames, or display them on a screen. In general, the components of an application are distributed over the nodes of a distributed system. Each component is bound to a thread and may be scheduled individually by the operating system of the corresponding node.

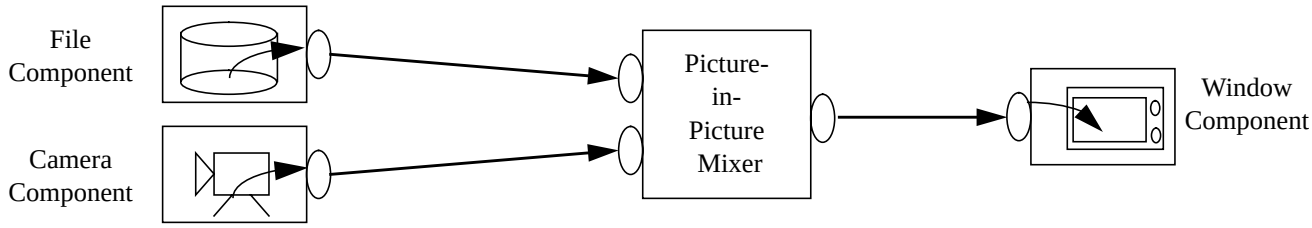

Figure 1: Picture-in-Picture Scenario

Fig. 1 depicts the processing topology of a picture-in-picture application. A camera component generates a live data stream. Another stream is read from file by a file component. The mixer component sets the frames of one stream as smaller images into the frames of the other. Finally, the picture-in-pictures frames are played out by a window component.

Components read and write data units of streams via so-called **ports**. Ports constitute location transparent data access points. Their interconnection allows to establish arbitrary processing topologies. It is distinguished between source, intermediate and sink components. A source component only generates data units by writing them to the output port(s). An intermediate component reads data units from one or many input ports, transforms them and writes the resulting data units to output ports. Sinks only consume data units via input ports. The focus of this paper is on the data access and periodic activation of intermediate and sink components. The further description assumes the following system architecture (see Fig. 2).

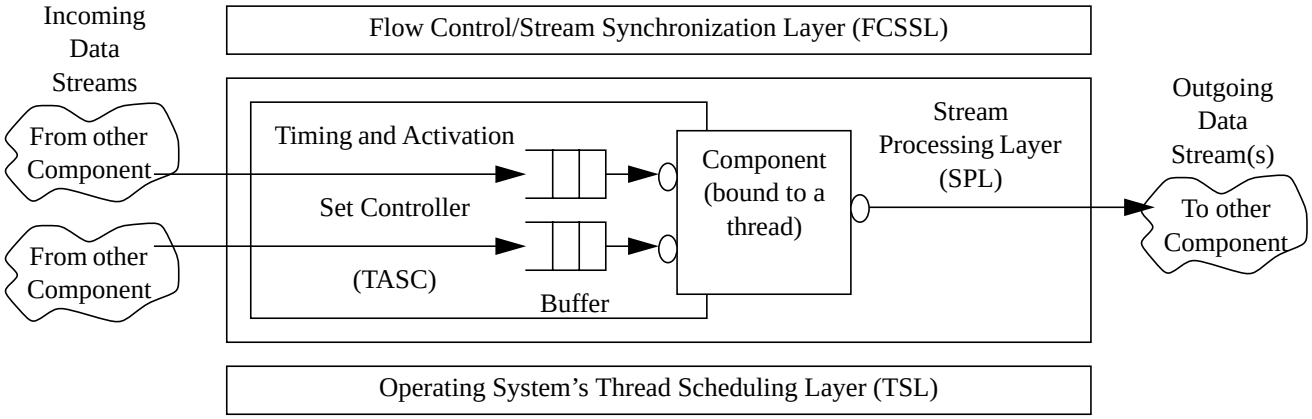

Figure 2: System Architecture for Timed Activations of Components

The system architecture for controlling the component's processing of data streams consists of three layers, namely the Flow Control/Stream Synchronization Layer (FCSSL), the Stream Processing Layer (SPL) and the operating system's Thread Scheduling Layer (TSL). Stream processing is performed in the SPL by a component which is bound to an operating system thread. The scheduling times and the amount of data that may be read from input ports are handled by the SPL's Timing and Activation Set Controller (TASC) (see below). The TSL allows for resource reservation of the CPU and schedules the activations of multiple, concurrent threads. As the activation of components is a purely local algorithm, the FCSSL provides the global coordination of multiple TASC's along a data path, even when multiple transport system hops are part of the data path. It initializes each TASC, controls buffer levels and performs recovery in case of errors, e.g. due to late data units. The details of the FCSSL, i.e. the flow control mechanisms and synchronization policies as well as the protocols to realize them, are beyond the scope of this paper.

The Timing and Activation Set Controller controls the **activation loop** of components, i.e. it controls the periodical activation of the component's stream processing code, and selects the data units belonging to activation sets. The activation loop comprises four stages. First, the scheduling time of the next activation is derived from the temporal properties of the processed data streams and the amount of data requested by the component. The scheduling time is given to the operating system's thread scheduler which blocks the component's thread until the appropriate point in time. Before the processing of the component is activated again, the set of requested data units, the so-called *activation set*, is released. For that, all data units being part of the activation set are selected under consideration of, among other criteria, synchronization relationships. They are marked and may be read by the component via ports. Additionally, an error code is set to inform the component and the FCSSL about incomplete activation sets. This provides for appropriate recovery actions.

The proposed architecture allows the component's stream handling code to be fairly simple. In particular, it is free of any real-time timing information concerning their activations. Timed data stream handling requires only three primitives: With the blocking call `TASC.GetAS`, the component requests to be blocked until the next activation time. Note that this is the only primitive which blocks the component's thread. Even if multiple time-dependent data streams are consumed, no multiple waiting states occur. `TASC.GetAS` returns when the activation set may be read from the component's ports by the non-blocking primitive `Port_X.GetData`. The call `Port_X.GetData` returns the next data unit from the buffer of the port `Port_X` as long as it belongs to the actual activation set (an error code otherwise). The component processes the data units and, in the case of an intermediate component, writes resulting data units to the output ports with the non-blocking call `Port_Y.PutData`. From there the data units are transferred to the buffer of the next component along the data path either by a local communication mechanism (e.g. by copying the reference to the data unit) or via a network.

The following example (in C++-like notation) depicts the stream processing code of the picture-in-picture mixer mentioned in Fig. 1. The code is executed in the activation loop as long as the application is active.

```
PictureInPictureMixer :: Activate () {
  ErrorCode = TASC.GetAS();           // block thread to wait for data units
  if ( ErrorCode == OK ) {
    frame1 = Inport1.GetData();       // read frames from input ports
    frame2 = Inport2.GetData();

    frame3 = pip_mix(frame1, frame2); // mix frames
    Outport.PutData(frame3);          // write result to output port

  } else {
    // ... error handling ...
  }
}
```

The correctness of activation sets depends on the semantics of the respective application. When releasing data units, the TASC has to observe synchronization relationships among data streams. They range from tightly related streams allowing no skew at all to loosely coupled streams tolerating a certain skew or streams that are not related at all. Furthermore, the amount of data which is released in each activation may not be fixed a priori. Different amounts of data per activation may result, for example, from non-matching rates of incoming streams.

The following section discusses the activation set model which allows to handle these requirements.

4. ACTIVATION SET

Activation sets divide the time lines of correlated, time-dependent data streams into segments. A segment contains all data units that are processed in a single activation of a component. Before the segmentation of data streams into activation sets is discussed in detail, the parameters describing the temporal properties of data streams, the so-called **media time systems** (MTS), and their synchronization relationships are introduced.

A data stream is an ordered sequence of data units $d_0, .. , d_n$. Each data unit d_i is associated with a **timestamp** $TS(d_i)$. The data unit that is processed first carries the timestamp $T_0 = TS(d_0)$. The sequence of timestamps spans a time axis, the so-called media time. The duration of data units of periodic streams are consecutive, i.e. the end of the duration of a data unit marks the beginning of the succeeding one [Herr91]. Furthermore, media time durations $\Delta = TS(d_{i+1}) - TS(d_i)$ are identical for all data units. Let t^D denote the (nominal) real-time difference between consecutive data units. Then, the data rate of the stream is $R = 1/t^D$ (see Fig. 3).

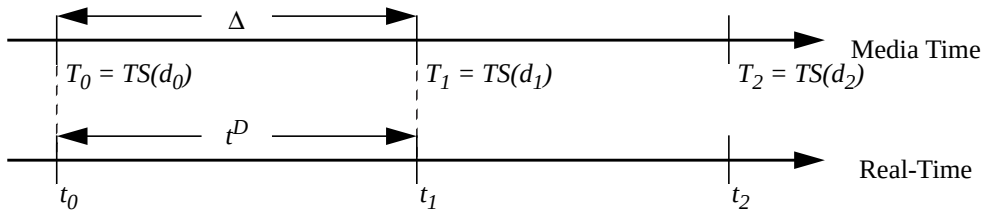

Figure 3: Media Time System MTS of a Stream

Synchronization relationships among media time systems are specified by so-called **reference points** (see Fig. 4). A reference point $RP = [MTS_1:T_1, MTS_2:T_2]$ defines that media time T_1 in media time system MTS_1 corresponds to media time T_2 in media time system MTS_2. In addition to this nominal synchronization relationship, a skew parameter $MTS_i:skew$ in either of both media time systems describes the tolerated deviation of the strict coupling of media time systems. Given the reference

point, media time can be transformed from one media time system into another. This is performed by a mapping function *TRANS* that maps media time m_1 of stream 1 onto media time m_2 of stream 2 as follows: $m_2 = TRANS(m_1) = T_2 + (m_1\text{-}T_1)\cdot\Delta_2/\Delta_1$.

In summary, the temporal properties of a data stream are described by the media time system $MTS = (T_0, \Delta, t^D)$, their synchronization relationships by reference points $RP = [MTS_1{:}T_1, MTS_2{:}T_2]$ and the maximum skew $MTS_i{:}skew$[1]. The media time systems of all streams and reference points describing their temporal relationships are supplied to the TASC by the FCSSL.

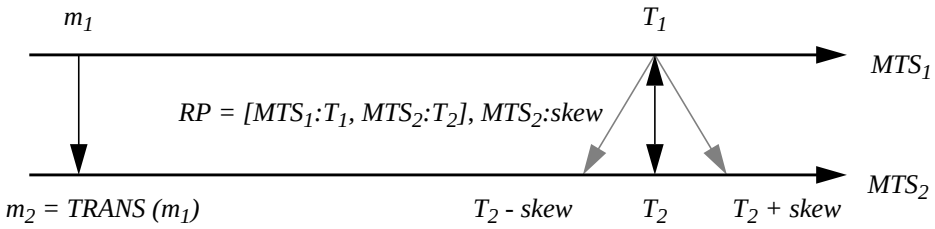

Figure 4: Synchronization Relationship among Media Time System

4.1. Segmentation of correlated media time systems into activation sets

Activation sets divide media time systems of data streams into segments. Like an ordinary data stream, activation sets have a nominal duration, denoted as L^M, which is a media time quantity (see Fig. 5). The duration of activation sets is specified relative to a dedicated stream, the so-called master stream. This avoids to introduce an additional, artificial time system for the description of temporal properties of activation sets. Furthermore, it allows to reduce the processing of fairly complex sets of data units to the processing of a periodic data stream, which simplifies the incorporation of the processing of activation sets into resource reservation protocols.

The component may specify the duration of an activation set and by that the amount of data it intends to consume in two ways. Either it states the duration L^M of the activation set directly as a media time quantity in the media time system of the master stream or by specifying the number k of data units it expects from the master stream. In the latter case, the duration of the activation set computes to $L^M = k \cdot \Delta_{Master}$. The latter approach offers the advantage that the component does not need to know the media time characteristics of the master stream.

From the duration of activation sets, their logical scheduling times may be derived. At the earliest, an activation set may be processed after its arrival, i.e. at $T_0 + L^M$ where T_0 corresponds to the timestamp of the first data unit of the master stream. In general, the i-th activation has a logical scheduling time of $T_0 + i\cdot L^M$. The mapping of logical scheduling times to real-time scheduling times is discussed in Section 5.

4.2. Data selection modes

The amount of data contained in an activation set is specified indirectly by its duration L^M. Whether or not a particular data unit is part of an activation set, is determined by the TASC just before the processing of the activation set begins. Depending on the tolerated skew among streams, the TASC supports several data selection modes, namely the *strict time-oriented*, the *relaxed time-oriented*, and the *set-oriented* selection mode. The selection mode that is to be applied may be set individually for each data stream.

In the *strict time-oriented* selection mode, timestamps are used to decide whether or not a data unit is part of an activation set. It does not permit any skew among data streams. An activation set has a duration of L^M. The i-th activation set begins at media time $T_0 + (i\text{-}1)\cdot L^M$ and ends at media time $T_0 + i\cdot L^M$ (exclusively). Both boundaries are media time quantities in the master

[1] For brevity, we assume a symmetric skew. An extension to an asymmetric skew (-x, +y) poses no problems.

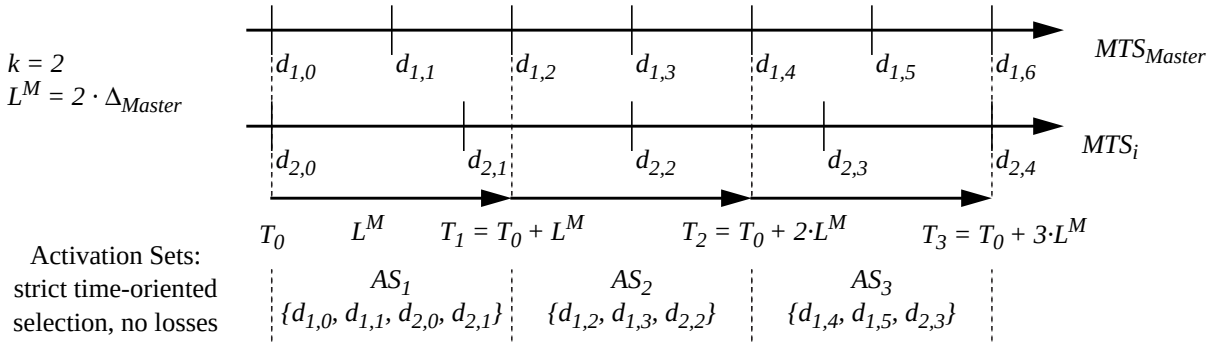

Figure 5: Activation Sets: Duration and Data Selection

stream's MTS. A data unit of the master stream is included in the *i*-th activation set, when its timestamp is between these media time boundaries. More precisely, the activation set of the *i*-th activation comprises any data unit d_j of the master stream whose timestamp $TS(d_j)$ satisfies the following inequation:

$$d_j \in AS_i \leftrightarrow T_0 + (i\text{-}1)\cdot L^M \leq TS(d_j) < T_0 + i\cdot L^M.$$

For data streams other than the master stream, selection of data units can only take place after having transformed the media time boundaries of the activation set from the MTS of the master stream into their media time systems, i.e. the application of the function *TRANS* is required. Thus, a data unit is part of an activation set, if its timestamp is between the transformed boundaries:

$$d_j \in AS_i \leftrightarrow TRANS\,(T_0 + (i\text{-}1)\cdot L^M) \leq TS(d_j) < TRANS\,(T_0 + i\cdot L^M)$$

Obviously, data selection is strict. In particular, a data unit that is late for its regular activation set due to, say, unexpected communication delays will never be included into another activation set. Such a data unit simply is discarded by the TASC upon detection. However, late or missing data units lead to *incomplete* activation sets, i.e. a component cannot be delivered the requested amount of data. The TASC's reaction is twofold: On the one hand, the error is signalled to the FCSSL. Its reaction depends on the implemented synchronization policy. It can, for example, increase the buffer levels by delaying further activations of the component to decrease the probability of future incomplete activation sets. Alternatively, it may try to renegotiate the transmission delay of the particular stream, or simply accepts the processing of incomplete activation sets as long as the number of errors is below a certain threshold. On the other hand, the component is informed about the incomplete activation set. Any error recovery, such as inserting a default data unit or reusing an older one, is performed inside the component. Contrary to other approaches, it was decided not to include any application-oriented error recovery into the TASC. A component best knows what to do in case of missing data units as it knows the semantics of its processing. Consequently, we can achieve a lean data access interface and an efficient implementation of the TASC.

Strict time-oriented selection realizes a synchronization relationship without any skew among data streams. The tolerance of human perception requires to support certain skew bounds between data streams which allows to reduce the necessity of discarding data units. Therefore, *relaxed time-oriented* selection allows for skew by widening the data selection window or shifting it against its regular position by (at most) the tolerated skew bound. Let *skew* denote the tolerated skew of a data stream against the nominal boundaries of the activation set. Then, a data unit may be part of the *i*-th activation set, if its timestamp satisfies the following inequation:

$$d_j \in AS_i \leftrightarrow TRANS\,(T_0 + (i\text{-}1)\cdot L^M) \pm skew \leq TS(d_j) < TRANS\,(T_0 + i\cdot L^M) \pm skew$$

The additional criterion is the amount of data units C_{max} that would be released in the error-free case without widening the selection window. When filling up incomplete activation sets, the TASC at most releases C_{max} data units. *Relaxed time-oriented* selection results in less discarded data units as late data units are processed in the succeeding activation set as long as the given

skew bounds are not surpassed. Should an activation set nevertheless be incomplete, i.e. even a widened selection window does not allow to fill up the activation set, signalling and error recovery are performed as mentioned above. Additionally, the way how skew among data streams is reduced, is depending on the synchronization policy. Thus, it is handled by the FCSSL which may, e.g., remove or duplicate data units in the buffers of ports.

In the *set-oriented* selection mode, only the amount of data but no timestamps are considered to decide the inclusion of data units in an activation set. The number of data units in an activation set is L^M/Δ for a particular stream. Either Δ is a multiple of L^M and the quotient is used directly, or remainders are summed up and considered accordingly. The selection mode allows to model an infinite skew and, additionally, it does not leave delayed data units out of consideration. If a data unit is not available at activation time due to being late, it simply is delivered in the next activation set. This selection mode is suited for components that process rather independent, i.e. non-synchronized, data streams.

After having introduced the data selection modes, we will illustrate their usage with several scenarios of the picture-in-picture mixing. Mixing of non-related data streams occurs when, say, the live transmission of a sports event is inset into the playback of a stored commercial. The stream of the commercial is master stream, its data units are selected by strict time-oriented selection to ensure best quality. The live data stream, besides being inset only as a small image, is not related to the commercial. Consequently, it may drift without causing recovery measures and is released in set-oriented mode. Slightly related contents may be observed when a news broadcast is combined with the translation of its contents into the “sign speech” that deaf-mutes use to communicate. Both streams should be mixed within certain skew limits to keep up the content’s relationship. Thus, the relaxed time-oriented selection mode is the appropriate choice. Strict time-oriented selection is more appropriate when mixing video streams of several cameras scanning the same area, e.g. in a surveillance application.

The data selection modes support the processing of aperiodic data streams, too. This is due to the fact, that the activation set concept is tolerant to missing data units as decisions are based on the specified media time systems and not on the actual existence of data units. An aperiodic data stream may simply be seen as a periodic data stream with a lot of missing data units. However, it is recommended not to use an aperiodic data stream as master stream to avoid unnecessary activations by deriving activation times from the (artificial) MTS of such a stream.

In summary, data selection is influenced by the duration of the activation set and the selection mode of each stream. The parameters, namely which stream is master stream and the relationship of the activation set’s duration to the duration of the master stream’s data units, are provided by the component. The streams’ selection modes are supplied by the FCSSL. In general, the parameters could be altered before each activation. However, our current implementation only supports a static definition of parameters when initializing the TASC. This is due to the fact, that changes in activation sets may lead to changes in resource requirements, e.g. in a modified activation rate of components. Static parameters allow omitting the renegotiation of resource reservations during run-time.

5. SCHEDULING OF THE PROCESSING OF ACTIVATION SETS

The section discusses the mapping of logical scheduling times of activation sets to real-time scheduling times. We give an overview of the interactions between the Stream Processing Layer and the Thread Scheduling Layer and discuss the computation of scheduling times in some detail. The discussion summarizes the integration of the activation set concept into the *CINEMA* multimedia environment [RBH94]. In this environment, we use a *split level scheduler* (see [GoAn91]). Scheduling of periodic real-time tasks is based on a *rate monotonic* algorithm which was enhanced for achieving shorter delays [Bart94].

In general, rate monotonic scheduling performs the execution of threads depending on a fixed priority. The priority of a task is derived from the rate of its activations. The higher the rate, the higher the priority. A thread is executed at the earliest at scheduling time *ST* (see Fig. 7). Its execution is finished at the latest at the scheduling time of the next processing step as long as some preconditions are met (see [LiLa73]). Among others, the execution time of a task has to be shorter than its period and the load of periodic tasks should not exceed approximately 69% of the overall capacity to avoid deadline violations. To reserve resources for processing of a periodic task, the rate and the execution time of a task is given to the TSL where a schedulability test is performed and, if successful, the required processing capacity is reserved. This either happens before start-up of an application or, in particular in the case of changes in activation sets, during run-time.

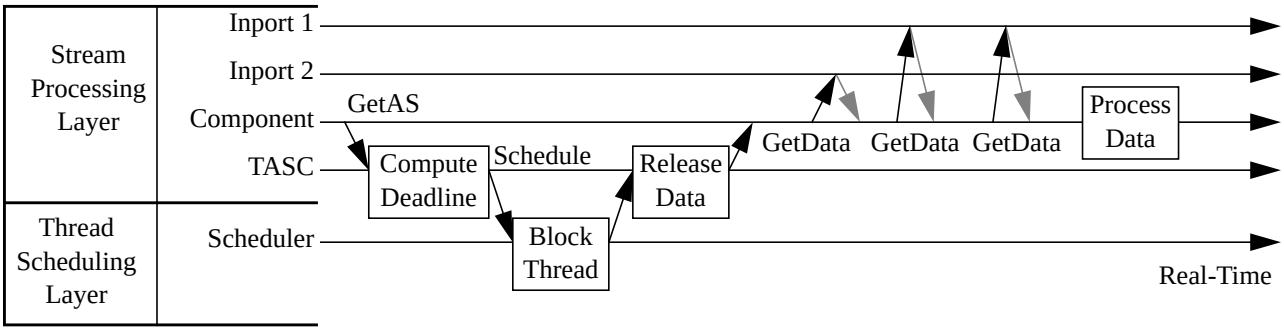

Figure 6: Scheduling of an Activation Set

The processing of a component is scheduled when it requests the next activation set AS_i by calling `GetAS` (see Fig. 6). The call is performed at call time CT_i immediately after the component finished the processing of activation set AS_{i-1}. `GetAS` causes the TASC to compute the scheduling time ST_i of the next (*i*-th) activation and to pass it to the thread scheduler. The scheduler blocks the thread at least until the scheduling time. As soon as the scheduler activates the thread again, the TASC selects the data units of the actual activation set (see Section 4.2) and the component's `GetAS` call returns. Then, the data units are ready to be read from the ports and can be processed immediately.

Computation of scheduling times depends on the activation rate and the execution time of components and on a possible offset provided by the FCSSL. An activation set has a nominal real-time duration L^D which is derived from the media time system of the master stream: $L^D = t^D \cdot L^M/\Delta$. The duration L^D determines the nominal activation rate $1/L^D$ of the component. Additionally, it provides the scheduling priority for rate-monotonic scheduling of the thread running the component code. The processing of an activation set takes an execution time *ET*. Thus, the latest possible start time for the processing of activation set AS_i, i.e. the deadline D_i, is $D_i = ST_{i+1} - ET$. The deadline may be expressed in relation to the real-time begin t_0 of the first data unit of the master stream (i.e. the data unit with the timestamp T_0): $D_i = t_0 + (i+1) \cdot L^D - ET + offset$. Note the additional offset, which allows the FCSSL to shift activation times to manipulate buffer levels. From the deadline D_i, the earliest possible activation time of the *i*-th activation, the scheduling time ST_i, is derived: $ST_i = D_i - (L^D - ET) + offset = t_0 + i \cdot L^D + offset$. Fig. 7 illustrates the scheduling of the *i*-th activation at latest possible time. The component is activated at the deadline D_i.[2] In contrast, the *(i-1)*-th activation starts prior to its deadline D_{i-1}.

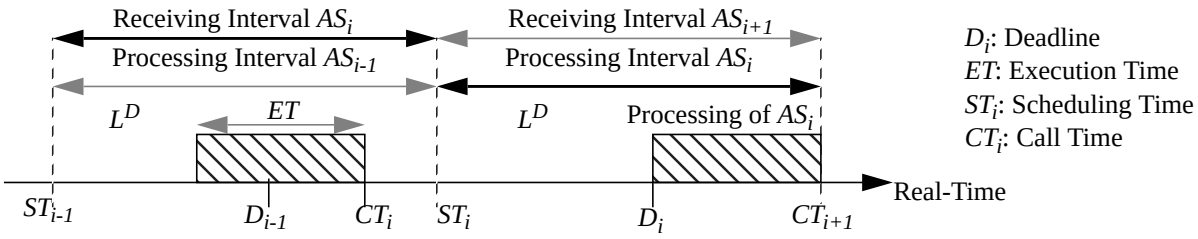

Figure 7: Scheduling Model

Overall, receiving and processing AS_i takes at most $2 \cdot L^D$ real-time units. This amount of time adds to buffering and transmission delays of data units. The scheduling time ST_i is the point in time when the arrival of the data units of activation set AS_i should be completed. Otherwise, data units are late if processing actually starts at the earliest possible time. Thus, $ST_i - ST_{i-1} = L^D$ real-time units are needed for arrival of the data units belonging to AS_i. At maximum another L^D units for scheduling and processing AS_i are required. The latter time may be reduced by certain modifications of the rate monotonic scheduling [Bart94].

[2] Note that, for simplicity, possible preemptions have not been taken into consideration.

Besides influencing the end-to-end delay, the duration L^D of an activation set has consequences for the size of input queues, too. The longer the activation set the more buffer space is needed - which adds to the buffers required for jitter elimination and delay equalization for synchronization purposes.

The computation of scheduling times wouldn't work without the initial value t_0. Therefore, let us assume a simple scenario to give an idea of how the FCSSL determines t_0. Given a source that is directly followed by a filter, we show how to compute t_0^F of the filter component. The filter consumes one data unit at each activation. Let d_{SF} denote the communication delay between the source and the filter component for one data unit. Assuming t_0^S to be given by the user, the scheduling time of the source's *i*-th activation component is $ST_i = t_0^S + i/R$. This means that the source claims *1/R* real-time units at maximum. Thus, the first data unit of the stream will appear at the latest at $t_0^F = t_0^S + 1/R + d_{SF}$ at the filter component. This value is computed by the FCSSL, which has a global view of the distributed application, and is passed to the TASC of the filter.

6. CONCLUSIONS

We have introduced the concept of activation sets as an abstraction for data access in distributed multimedia systems. It provides for controlling the timely processing of sets of data units. Components are supported that consume multiple data streams which may have different rates and are related by more or less stringent synchronization relationships. Scheduling times are derived from temporal properties of streams and passed to a modified rate monotonic scheduler. The approach supports scaling transparency, separates component programming from the timing of their activations and integrates transport and operating system functions by providing a seamless transition from releasing data units at the edge of a transport system to their actual processing scheduled under control of the operating system.

The described activation set concept assumes components to be passive. To integrate components having an internal timing, e.g. a component that controls an audio board which is triggered by its hardware timer, some modifications are required. The activations of such components are not data driven. Instead, these components signal their readiness for activation themselves. The components may request the selection of data units from the TASC by a non-blocking variant of the `GetAS` call. Data units are selected according to the actual real-time. Note however, that as soon as real-time periods are provided by the component the advantage of scaling transparency gets lost.

Currently, we are implementing the activation set concept in the *CINEMA* multimedia environment to gain more practical experiences. Future work includes the in-depth discussion of the activation of components having an internal timing and the extension of the activation set concept by event-driven activations, e.g. to support a step-by-step execution of stream handlers under control of the application for debugging purposes. Additionally, the incorporation of "work-ahead-strategies" into the TASC are investigated as well as the support of alternative master streams in the case of groups of streams having the same temporal properties.

7. REFERENCES

[Appl91] Apple Computer Inc., Cupertino, CA, USA. *QuickTime Developer's Guide*, 1991.

[BaLe94] Fabio Bastian and Patrick Lenders. Media Synchronization on Distributed Systems. *IEEE Intl. Conference on Multimedia Computing, Boston*, 5 1994.

[Bart94] Ingo Barth. Extending the Rate Monotonic Scheduling Algorithm to Get Shorter Delays. *International Workshop on Advanced Teleservices and High-Speed Communication Architectures, Heidelberg, Germany*, 9 1994.

[BDH+93] Ingo Barth, Gabriel Dermler, Tobias Helbig, Kurt Rothermel, Frank Sembach, and Thomas Wahl. CINEMA: Eine konfigurierbare, integrierte Multimedia-Architektur. In Kurt Rothermel Wolfgang Effelsberg, editor, *Verteilte Multimedia Systeme, Tagungsband GI/ITG-Arbeitstreffen, Stuttgart*, pages 33–47. Saur Verlag, 1993.

[CBRS93] Geoff Coulson, Gordon S. Blair, Philippe Robin, and Doug Shepherd. Extending the Chorus Micro-Kernel to Support Continuous Media Applications. In *Proceedings of the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 49–60, 11 1993.

[DeKr75] F. DeRemer and H. Kron. Programming-in-the-large versus programming-in-the-small. In *Proceedings of the Conference on Reliable Software*, pages 114–121, 1975.

[DNNR92] Roger B. Dannenberg, Tom Neuendorffer, Joseph M. Newcomer, and Dean Rubine. Tactus: Toolkit-Level Support for Synchronized Interactive Multimedia. In *3nd International Workshop on Network and Operating System Support for Digital Audio and Video*, 11 1992.

[EDP92] Julio Escobar, Debra Deutsch, and Craig Partridge. Flow Synchronization Protocol. In *IEEE Global Communications Conference*, 12 1992.

[GBT94] Simon Gibbs, Christian Breiteneder, and Dennis Tsichritzis. Data Modeling of Time-Based Data. *Proceedings SIGMOD '94*, 1994.

[Gibb91] Simon Gibbs. Composite Multimedia and Active Objects. *Proceedings OOPSLA '91, Phoenix, Arizona*, pages 97–112, 10 1991.

[GoAn91] R. Govindan and D. P. Anderson. Scheduling and IPC Mechanisms for Continuous Media. *OS Review*, 25(5):68–80, Oct. 1991.

[Herr91] Ralf Guido Herrtwich. Time Capsules: An Abstraction for Access to Continuous-Media Data. *The Journal of Real-Time Systems, Kluwer Academic Publishers*, pages 355–376, 3 1991.

[Hewl93] Hewlett-Packard Company and International Business Machines Corporation and SunSoft Inc. *Multimedia System Services, Version 1.0*, 7 1993.

[KrMa85] Jeff Kramer and Jeff Magee. Dynamic Configuration for Distributed Systems. *IEEE Transaction on Software Engineering*, SE-11(4):424–436, 4 1985.

[LiLa73] C.L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1 1973.

[LKG92] Li Li, A. Karmouch, and N.D. Georganas. Real-time Synchronization Control in Multimedia Distributed Systems. In *Multimedia '92, 4th IEEE ComSoc International Workshop on Multimedia Communications*, 4 1992.

[MKSD90] Jeff Magee, Jeff Kramer, Morris Sloman, and Naranker Dulay. An Overview of the REX Software Architecture. In *2nd IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, 10 1990.

[NaSm92] Klara Nahrstedt and Jonathan M. Smith. The Integrated Media Approach to Networked Multimedia Systems. 2 1992.

[RaBa93] K. Ravindran and Vivek Bansal. Delay Compensation Protocols for Synchronization of Multimedia Data Streams. *IEEE Transactions on Knowledge and Data Engineering, Vol. 5, No 4*, pages 574–589, 8 1993.

[RBH94] Kurt Rothermel, Ingo Barth, and Tobias Helbig. *Architecture and Protocols for High-Speed Networks*, chapter CINEMA - An Architecture for Distributed Multimedia Applications, pages 253–271. Kluwer Academic Publishers, 1994.

[RoHe94] Kurt Rothermel and Tobias Helbig. ASP: Adaptive Synchronization Protocol for Continuous Data Streams. *Technical Report 14/94, University of Stuttgart/IPVR*, 12 1994.

[RRVK92] P. Venkat Rangan, Srinivas Ramanathan, Harrick M. Vin, and Thomas Käppner. Media Synchronization in Distributed Multimedia File Systems. In *Proceedings of the 4th IEEE ComSoc Int. Workshop on Multimedia Communications, Monterey (CA), USA*, pages 315–324, 4 1992.

[RVR93] P. Venkat Rangan, Harrick M. Vin, and Srinivas Ramanathan. Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences. *IEEE/ACM Transactions on Networking, Vol. 1, No. 1*, pages 20–30, 2 1993.

[ShSa90] Shepherd and Salmony. Extending OSI to Support Synchronisation Required by Multimedia Applications. *Computer Communications, 7(13):399-406*, 9 1990.

[SZS94] Robert Simon, Taieb Znati, and Robert Sclabassi. Group Communication in Distributed Systems. *14th ICDCS, Poznan, Poland*, 5 1994.