On the Correctness of Multimedia Applications

Erwan Demairy   Emmanuelle Anceaume   Valérie Issarny
IFSIC   CNRS   INRIA
IRISA - Solidor, Campus de Beaulieu, 35042 RENNES Cedex (France).
Fax: 33 2 99 84 25 29

E-mail: {`Erwan.Demairy, Emmanuelle.Anceaume, Valerie.Issarny`}`@irisa.fr`

Abstract

This paper provides a method to verify that the properties we consider critical to build a correct multimedia application are met. These properties are the compatibility of the protocols employed by the computing elements to interact with each other and the timeliness of the application. The method relies on (i) an original model that enables a software architect to specify the basic elements of its multimedia application according to the two aforementioned properties, and (ii) two algorithms that determine the best level of perceived quality of service that can be guaranteed by the application.

1. Introduction

A multimedia application is characterized by its ability to handle quasi-periodic streams. Typically, these streams transport a large amount of data (*e.g.* video or audio streams) and have soft temporal constraints, *i.e.* missing a deadline does not have any critical consequences for users of the application. In general, temporal constraints apply either on a single stream or on several ones, which requires synchronization of the different streams.

In this paper, we focus on the description of software architectures of multimedia applications. The software architecture research domain has produced different notations and associated tools that are proven to be useful for the design and construction of applications [4]. Existing work in the area address both the definition of notations for the specification of the properties the software architect focuses on (*e.g.* functional behavior), and tools to verify the behavior of an application built out of existing parts.

However, existing environments providing support for the design of applications based on their architectural description lack expressiveness for two issues raised by multimedia applications: the compatibility of protocols used for transferring data streams (*e.g.* specialization) and the verification of the application timeliness. The second issue has been addressed in [8, 1]. However, their solutions rely on mathematical models that are either too formal or too empirical. In [8], the temporal behavior of the application is computed only by relying on formal distributions. While this method is easily applicable to a limited set of distributions, it turns out to be a tedious, even impossible task when arbitrary formal distributions need to be composed. A convincing example is given by the determination of the distribution of the maximum of any two arbitrary normal distributions. The mathematical model proposed in [1] relies on a simulation of the temporal behavior of the application and thus leads to the opposite problem. This model requires that whenever a group of elements is reused, the temporal verification tool has to simulate the global behavior of the application on a per element basis.

In this paper, we consider both the specification and the verification of the two aforementioned properties of multimedia applications: a formal framework provides means to the software architect to specify the protocols and temporal behavior of the basic elements of his application. According to these specifications, the verification process determines the set of protocols that are eligible for carrying out interactions among computing elements based on the data streams that are exchanged. This verification is made of three steps:

(i) the first step determines whether the protocols of the application are consistent or not. This step results in identifying the protocols that can be used by each module of the application according to the interaction patterns among the application's modules;

(ii) for each combination of the protocols allowable for the application modules, the temporal behavior of the application is compared with its temporal constraints. The temporal behavior can either be formally computed or simulated, depending on the knowledge the

This research was supported by the C3DS project under contract ESPRIT Long Term Research Project 24962

software architect has of the application. This step results in determining whether the application is consistent or not according to its temporal behavior;

(iii) the two previous steps assess the correctness of the multimedia application both in terms of protocol usage and timeliness properties. If the application is correct, there exists a non-empty set of protocols guaranteeing its correctness. To configure the application, we choose one of the maximal elements of this set according to a user-satisfiability ordering relationship.

Thus, we provide a method that alleviates the work of the software architect by allowing him to verify *a priori* (*i.e.* without building the application) the feasibility of his application. Furthermore, our method allows a hierarchical description of an application, which permits its progressive development. Consequently, adding a new element to a formerly assessed group of elements amounts to compute the algorithms for an architecture composed of the group taken as an element and the new element.

The remainder of this paper is organized as follows. Section 2 gives an overview of the main features pertaining to multimedia applications software architecture. An example of a virtual studio application illustrates our work. Section 3 presents the algorithm checking the compatibility of communication protocols. The algorithm verifying its temporal correctness is described in Section 4. Perspectives of this work conclude this paper.

2. Multimedia Application Style

According to the terminology of the software architecture research domain [7], the *style* of a multimedia application is the set of rules that permit to determine whether an application can be considered as a multimedia application or not. These rules encompass the eligibility of the computing and communication elements according to some criteria, *e.g.* constraints imposed by the target execution environment.

A multimedia application is composed of two kinds of *elements*. The former represents the computing elements of the application, which are called *components*; the latter are in charge of transmitting data (*e.g.* local or wide area network), and are called *connectors*.

The interaction between a component and a connector is done by a *port*. Ports are used to send or receive streams of data frames. We further call *server* any component that produces data frames and *client* any component consuming these frames. Without loss of generality, we assume that the time between any two message transmissions within a stream follows a probabilistic distribution. This restriction allows us to model applications in which data exchange follows a continuous temporal behavior over long periods of time, *e.g.* VoD and tele-medicine applications.

The server of a data stream usually performs on it some coding for transmission (*i.e.* protocols encoding), cost (*i.e.* compression) or security reasons (*i.e.* encryption). At the end point of the stream, the software architect has to ensure that the target client is able to decode the data stream by using a decoding process compatible with the coding one. These transformations are carried out according to specific protocols [9], whether they concern audio stream (*e.g.* PCM), video stream (*e.g.* CCIR-601, MPEG-2), or arbitrary bit stream (*e.g.* `TCP/IP`). This latter type of protocol is denoted in the following by "`any`".

Beyond its ability to handle a specific coding or decoding process, a protocol is characterized by its temporal behavior (Section 4). In general, the elements of a multimedia application handle several protocols, in order to adapt the quality of their output regarding available resources (*e.g.* CPU, bandwidth). Thus, when building an application out of existing elements, we need to determine which protocol must be used for each element of the application. This paper solves this problem.

Throughout the paper, we illustrate our proposal by verifying whether a virtual studio application complies with the multimedia style. The architecture of the application is depicted in Figure 1. The virtual studio application [3] consists of mixing a virtual background and a real foreground (*e.g.* the speaker) in real-time. In this example, real pictures are taken by a camera linked to a computer, symbolized by component C_1. In parallel, the virtual background is calculated by component C_2 by using the position of the camera into the scene. Finally, the two streams of pictures are mixed in C_3. Component C_3 outputs MPEG-2 and CCIR-601 video streams (*i.e.* with or without compression) and transmits it to the client components C_5 (the central control room), C_6 (monitor screen of the monitor), and C_7 (individual TV).

An accurate description of the protocols used by each element of a multimedia application is central to the verification of the application correctness. From this description, given by the software architect, we first determine the protocols eligible for each element of the application, by considering the constraints that each element implies on the others. Then, we verify among the eligible protocols which ones guarantee that the application meets its temporal constraints.

3. Protocols Consistency

This section begins by introducing ordering relationships on the protocols used by the application. Then, we address the issue of the compatibility among the elements of the application.

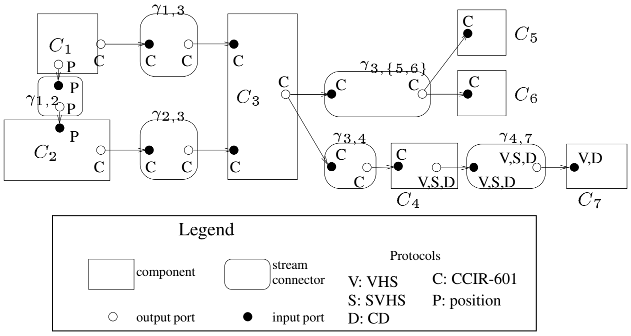


Figure 1. Virtual Studio Application

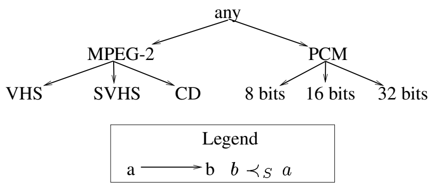


Figure 2. Specialization Relationship $\prec_S$

3.1. Protocol Relationships

An architectural element E has N_E^i input ports noted $i_E^1, \ldots, i_E^{N_E^i}$ and N_E^o output ports noted $o_E^1, \ldots, o_E^{N_E^o}$. To simplify these notations, we omit superscript when we focus on a single port.

Each port of an element supports at least one protocol. We denote by π_p the set of protocols supported by a port p. The capability of a port to support more than one protocol comes from the multiplication of adaptive multimedia applications together with available frameworks supporting adaptive services, sometimes called Quality of Service (QoS) Architectures [5, 2]. The *state* of a port p is the value s_p taken in the set π_p. It indicates the protocol employed by a port at a given time.

Ordering Relationships. We introduce two partial ordering relationships. The *specialization relationship* is used to determine whether two elements can interact based on the protocols supported by their connected ports. The *Perceived QoS relationship* characterizes the best service that may be provided to the user. Both relationships are detailed thereafter.

A protocol a is *compatible* with a protocol b, noted $a \prec_S b$, if b is reducible to a. For example, in Figure 2, protocol `VHS` (cf. [2]) is a specialization of `MPEG-2`, which is itself a specialization of protocol `any`. The expression $a =_S b$ states that a and b are the same protocol.

Given that a port can support several protocols, any two elements bound by two ports may interact by using compatible protocols. In order to enforce the selection of the best pair of protocols from the end-user's standpoint, we define a partial relationship, based on the QoS perceived by the user. This relationship is subjective (*i.e.* it is up to the users to define it [5]), and it is up to the software architect to specify this relation over available protocols. Figure 3 illustrates this relationship: `VHS` has a lower perceived QoS than `SVHS`, itself lower than `CD`. We note $a \prec_{QoS} b$ when a has a lower perceived QoS than b; the expression $a =_{QoS} b$ states that a and b provide the same QoS to the user.

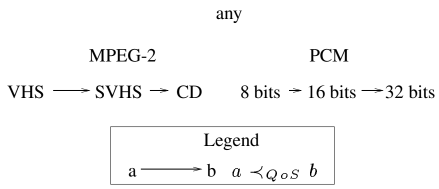


Figure 3. Ordering Relationship $\prec_{QoS}$

Further Definitions. In the following, the generic notation $a \preceq_? b$ states that either $a \prec_? b$ or $a =_? b$, with $? \in \{S, QoS\}$. Notations $a \Leftrightarrow_? b$ and $a \not\Leftrightarrow_? b$ respectively denote that protocols a and b are comparable (respectively not comparable) with respect to the "?" partial ordering relationship.

We extend the previous notations to compare sets of protocols. The relation $\pi_E \Leftrightarrow_? \pi_{E'}$ means that for each element s in π_E and t in $\pi_{E'}$, s and t are comparable for the used partial ordering relationship. In the same manner, $\pi_E \not\Leftrightarrow_? \pi_{E'}$ means that there exists at least one element s in π_E and one element t in $\pi_{E'}$ such that $s \not\Leftrightarrow_? t$. We extend the compatibility relationship to the sets of protocols: let π_E and π'_E be two sets of protocols. The relation $\pi_E \succ_S \pi'_E$ means that $\forall s \in \pi_E, \forall t \in \pi'_E, s \succ_S t$.

We extend the state of an individual port to the state of an element. The state of an element is the set of states of the ports that compose this element. By extension, the state of an application is the set of states of the different elements (*i.e.* components and connectors) composing the application. Notice that the $\prec_{QoS}$ relationship induces a natural partial ordering relationship on the states of the elements of the application. More formally, let S_E and S'_E be two possible states of the element E noted: $S_E = ((s_{i_E^1}, \ldots, s_{i_E^n}), (s_{o_E^1}, \ldots, s_{o_E^q}))$ and $S'_E = ((s'_{i_E^1}, \ldots, s'_{i_E^n}), (s'_{o_E^1}, \ldots, s'_{o_E^q}))$ then $S_E \preceq_{QoS} S'_E$ if and only if we have $s_{i_E^k} \preceq_{QoS} s'_{i_E^k}$ and $s_{o_E^l} \preceq_{QoS} s'_{o_E^l}$ for

$1 \leq k \leq n$ and $1 \leq l \leq q$. This relationship orders the states of the application to select the best one from the user's perspective: the best state is taken within the set of maximal states of the application for the $\prec_{QoS}$ relationship.

The software architect defines for each element E of the application the constraints C_E on the states that can be simultaneously taken by its ports.

For example, the states that can be taken by the elements of the virtual studio application are depicted in Figure 4. The formulae must be understood as follows. For example, constraint C_{C_7} states that component C_7 handles two states $((\mathtt{VHS}), \emptyset)$ and $((\mathtt{CD}), \emptyset)$, where $((\mathtt{CD}), \emptyset)$ expresses that the single input port of C_7 accepts `any.MPEG-2.CD` protocol, and that there is no output port ($\emptyset$). The protocol `position` stands for the data related to the position of the camera that are sent from the camera (C_1) to the computer (C_2). The empty set ($\emptyset$) is used whenever there is no input or output port associated to a given element.

$C_{C_1} = \{(\emptyset, (\mathtt{CCIR\text{-}601,position}))\}$ $\quad C_{C_2} = \{((\mathtt{position}), (\mathtt{CCIR\text{-}601}))\}$

$C_{C_3} = \{((\mathtt{CCIR\text{-}601,CCIR\text{-}601})(\mathtt{CCIR\text{-}601}))\}$ $\quad C_{C_5} = \{((\mathtt{CCIR\text{-}601}), \emptyset)\}$

$C_{C_6} = C_{C_5}$ $\quad C_{C_7} = \{((\mathtt{VHS}), \emptyset), ((\mathtt{CD}), \emptyset)\}$

$C_{\gamma_{1,2}} = \{((\mathtt{position}), (\mathtt{position}))\}$ $\quad C_{\gamma_{1,3}} = \{((\mathtt{CCIR\text{-}601}), (\mathtt{CCIR\text{-}601}))\}$

$C_{\gamma_{2,3}} = C_{\gamma_{3,\{5,6\}}} = C_{\gamma_{3,4}} = C_{\gamma_{1,3}}$

$C_{C_4} = \{((\mathtt{CCIR\text{-}601}), (\mathtt{VHS})), ((\mathtt{CCIR\text{-}601}), (\mathtt{SVHS})), ((\mathtt{CCIR\text{-}601}), (\mathtt{CD}))\}))$

$C_{\gamma_{4,7}} = \{((\mathtt{VHS}), (\mathtt{VHS})), ((\mathtt{SVHS}), (\mathtt{SVHS})), ((\mathtt{CD}), (\mathtt{CD}))\}$

Figure 4. Studio Application Constraints

Among the states allowable for each element of an application, only a subset of them are compatible with the states of the other elements of the application. Thus, we need a method to compute the eligible states of a multimedia application according to the protocol compatibility property.

3.2. Verifying Protocol Compatibility

The eligible states of an application depend on the constraints that each element imposes on the elements it is bound to.

Let us illustrate the first step of the verification process by considering a couple of elements. Such a configuration is simple, since the constraints may directly be compared.

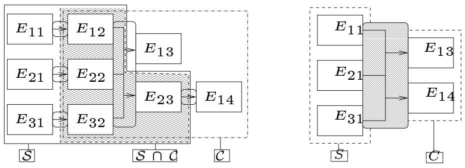
Connection of Two Ports. Let E_1 and E_2 be two elements. The connection of port o_{E_1} to port i_{E_2} requires to build the set of possible protocols accepted by this connection, *i.e.* the set of data encoding compatible with both o_{E_1} and i_{E_2}. This compatibility is denoted by the relation $\pi_{o_{E_1}} \triangleright \pi_{i_{E_2}}$. For each state s in $\pi_{o_{E_1}}$, there are three possibilities:

- $\forall t \in \pi_{i_{E_2}}, t \prec_S s$: connecting these two ports is not possible because the protocols of i_{E_2} are too specialized regarding to protocol s. Thus $\{s\} \triangleright \pi_{i_{E_2}} = \emptyset$.
- $\exists t \in \pi_{i_{E_2}}, s \preceq_S t$: protocol s is more specialized than $\pi_{o_{E_2}}$. To preserve the actual encoding of the incoming data. Then $\{s\} \triangleright \pi_{i_{E_2}} = \{s\}$.
- $\{s\} \not\Leftrightarrow_S \pi_{i_{E_2}}$: the connection is not correct and there is no compatibility between the two ports. Thus $\{s\} \triangleright \pi_{i_{E_2}} = \emptyset$

This leads to the expression $\pi_{o_{E_1}} \triangleright \pi_{i_{E_2}} = \{s \in \pi_{o_{E_1}}, \exists t \in \pi_{i_{E_2}}$ with $s \preceq_S t\}$.

To represent the state eligible for an element when the connection involves several ports, we extend this notation to the multiple connection of N sets of protocols:

$$\mathop{\times}_{i=1}^{N} \pi_i \triangleright \mathop{\times}_{i=1}^{N} \pi'_i = \mathop{\times}_{i=1}^{N} \pi_i \triangleright \pi'_i$$



(a) "Complex" View (b) Simplified View

Figure 5. Simplification of an application

Connection of n Elements. Let us extend our results to n elements. Our method consists in computing the set of possible states of the application in a hierarchical way. More precisely, the behavior of a set of connected elements is expressed only by considering the states of its input and output elements.

Let $\mathcal{S}$ be the set of elements with output ports, named *server elements* and let $\mathcal{C}$ be the set of elements with input ports, named *client elements*. A complex view of an application where all the elements are described (Figure 5(a)) can be reduced to a simplified view (Figure 5(b)), where only server-only components ($\mathcal{S}_{only} = \mathcal{S} \backslash \mathcal{S} \cap \mathcal{C}$), and client-only elements ($\mathcal{C}_{only} = \mathcal{C} \backslash \mathcal{S} \cap \mathcal{C}$) are considered.

We propose an algorithm computing the possible states of each element depending on both the constraints of its predecessors (*i.e.* bound to it through its input ports) and the constraints of its successors (*i.e.* bound to it through its output ports). This algorithm (called "propagation algorithm" in the following) is made of two parts. The first part of the propagation algorithm consists in propagating the constraints of input elements towards output elements. The second part consists in the reverse propagation. Let

$p(E)$ (respectively $s(E)$) be the sets of elements immediately preceding (respectively succeeding) element E.

Propagation from Inputs to Outputs. In this step, we compute the states that can be taken by each element regarding the constraints imposed by its predecessors. This is what we call a propagation of the constraints from inputs to outputs. Let E be an element such that $p(E) = \{E_1, \ldots, E_n\}$ with $n \geq 0$. Let $*$ denote any protocol. Let $\mathcal{I}_E^o$ be the acceptable states for an element E given the constraints of all its predecessors. Then, we have the following formula:

$$\mathcal{I}_E^o = \left(\mathbf{o} \left(\begin{array}{c} n \\ \bullet \\ i=1 \end{array} \mathcal{I}_{E_i}^o \right), * \right) \triangleright C_E \tag{1}$$

where the operator $\bullet$ represents the concatenation of two states, *i.e.* :
$((s_{i_E^1} \ldots s_{i_E^p})(s_{o_E^1} \ldots s_{o_E^q})) \bullet ((s_{i_F^1} \ldots s_{i_F^t})(s_{o_F^1} \ldots s_{o_F^u})) = ((s_{i_E^1} \ldots s_{i_E^p} s_{i_F^1} \ldots s_{i_F^t})(s_{o_E^1} \ldots s_{o_E^q} s_{o_F^1} \ldots s_{o_F^u}))$ and $\mathbf{o}$ denotes the projection of a state on its output ports, *i.e.* :

$$\mathbf{o}((s_{i_E^1}, \ldots, s_{i_E^n}), (s_{o_E^1}, \ldots, s_{o_E^q})) = (s_{o_E^1}, \ldots, s_{o_E^q})$$

Equation 1 is solved by recursion on parameter E. We begin to compute $\mathcal{I}_E^o$ for the elements of $S_0 = \mathcal{S}_{only}, \ldots, S_k = \left\{ E \in (\mathcal{S} \bigcup \mathcal{C}) / p(E) \in \bigcup_{i=0}^{k-1} S_i \right\}, \ldots, S_N = \mathcal{C}_{only}$.

By applying the propagation algorithm on the constraints of the virtual studio application (Figure 4), we obtain that $\forall E, \mathcal{I}_E^o = C_E$.

Propagation from Outputs to Inputs. Once $\mathcal{I}_E^o$ expressions have been computed for all the architectural elements, we propagate constraints upon the states towards input elements. Let $\mathcal{O}_E^i$ be the possible states for an element E by taking into account the constraints of all its successors. The possible states $\mathcal{O}_E^i$ of an element E such that $s(E) = \{E_1, \ldots, E_n\}$, with $n \geq 0$, are given by the formula:

$$\mathcal{O}_E^i = \left(*, \mathbf{i} \left(\begin{array}{c} n \\ \bullet \\ i=1 \end{array} \mathcal{O}_{E_i}^i \right) \right) \triangleright \mathcal{I}_E^o \tag{2}$$

Similarly to the propagation from input towards outputs, we solve this equation by recursion on E and iterate the computation on sets $T_0 = \mathcal{C}_{only}, \ldots, T_k = \left\{ E \in (\mathcal{S} \bigcup \mathcal{C}) / s(E) \in \bigcup_{i=0}^{k-1} T_i \right\}, \ldots, T_N = \mathcal{S}_{only}$.

The finite number of elements implies that the number of steps to make the computation is finite.

Once this computation is finished, the set of all the elements can be represented as a single element, having for inputs those of server-only elements, and for outputs those of client-only elements. For example, in Figure 5(a), components E_{12}, E_{22}, E_{32}, E_{23} and the associated connectors, may be grouped together as a single element (Figure 5(b)).

By applying the propagation algorithm on the constraints of the virtual studio application (figure 4) and on the values of $\mathcal{I}_E^o$ previously computed, we obtain that

$$\mathcal{O}_E^i = C_E \tag{3}$$

with $E = C_1, C_2, C_3, C_5, C_6, \gamma_{1,2}, \gamma_{1,3}, \gamma_{2,3}, \gamma_{3,\{5,6\}}, \gamma_{3,4}$

$$\left\{ \begin{array}{lcl} \mathcal{O}_{C_4}^i & = & \{((\texttt{CCIR-601}), (\texttt{VHS})), \\ & & ((\texttt{CCIR-601}), (\texttt{CD}))\} \\ \mathcal{O}_{\gamma_{4,7}}^i & = & \{((\texttt{VHS}), (\texttt{VHS})), ((\texttt{CD}), (\texttt{CD}))\} \\ \mathcal{O}_{C_7}^i & = & C_{C_7} \end{array} \right. \tag{4}$$

Let us examine these results. During the verification of the compatibility of the protocols, the virtual studio application can be divided into two parts: one is made of the local part of the application (C_1,C_2,C_3,C_5, C_6,$\gamma_{1,2}$,$\gamma_{1,3}$,$\gamma_{2,3}$,$\gamma_{3,\{5,6\}}$,$\gamma_{3,4}$), and the second part groups the MPEG-2 encoder and the TV set ($C_4, \gamma_{4,7}, C_7$). Unlike the former part in which no checking is necessary since only the `CCIR-601` protocol is used by all its elements (implying the result expressed in (3)), the consistency of the latter part depends on the protocols accepted by the TV set C_7. Element C_7 only supports `CD` and `VHS` protocols. Thus, it implies that the MPEG encoder C_4 has only to provide these protocols, resulting in equations (4). Thus, the protocols employed by the different components and connectors are compatible, because $\forall E, \mathcal{O}_E^i \neq \emptyset$. *A fortiori*, the virtual studio application is correct with respect to the protocol compatibility property.

When temporal dependencies lead to cycles, the propagation algorithm cannot be applied anymore. It is then possible to "linearize" the application as illustrated on Figure 6. Due to space limitation, proof of the correctness of the removal of cycles is not provided in this paper.

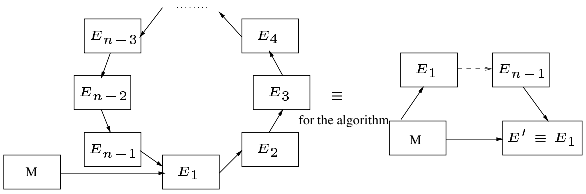


Figure 6. Simplification of Cycles

4. Timeliness Consistency

Server elements produce data by guaranteeing a given temporal behavior. In contrast, client components require the respect of temporal constraints on the arrival of data. This section is divided into two parts: the first describes the temporal specifications, and the second addresses temporal verifications regarding elements constraints.

4.1. Temporal Specification

Servers express temporal guarantees on time between (i) two successive output of frames on a given port and (ii) the input of a frame and the output of the resulting data. These informations are mandatory when dependencies exist between input and output ports within a component. Temporal guarantees can be provided either through the provision of a statistical sample of measurements or through the formal distribution describing the service times that are usually computed after a statistical analysis of the samples obtained by running the component. By comparison with other works (*e.g.* [8, 1]), the main advantage of our approach is that it emphasizes the reuse of group of elements as a single element. Thus, it reduces the amount of computations for the subsequent temporal verifications. Furthermore, it permits to obtain an overview of a set of elements. This enables the software architect to focus on the global temporal behavior of the group of element he uses, instead of having to handle all the details of the application.

Clients express temporal constraints on the distribution of the time between arrivals of frames. These constraints express the behavior that must be exhibited by the application. Let $\mathbf{P}(A)$ be the probability of event A. Temporal constraints may be expressed by three inequations:

$$\mathbf{P}(\Delta_{i_E^k} \in [min, max]) \geq \beta \qquad \text{(A)}$$

The expression $\Delta_{i_E^k}$ stands for the time between two successive inputs of a frame. This probability expresses that the constraints on the time between two inputs on port i_E^k must be fulfilled at least with a probability β.

$$\mathbf{P}(\max_{k,l \in \{\eta_1,\ldots,\eta_m\}} |\delta_{i_E^k} - \delta_{i_E^l}| \leq K) \geq \beta \qquad \text{(B)}$$

Variables $\delta_{i_E^k}$ and $\delta_{i_E^l}$ stand for the arrival time of the data on input ports i_E^k and i_E^l respectively, with k and l taken in $\{\eta_1, \ldots, \eta_m\}$, a set of indices of the input ports of E. This probability expresses that the maximum jitter between several streams on input ports $i_E^{\eta_1}, \ldots, i_E^{\eta_m}$ must be lower than K with a probability greater or equal to β.

$$\mathbf{P}(|\delta_{p_E} - \delta_{p_{E'}}| \leq K) \geq \beta \qquad \text{(C)}$$

This inequation expresses constraints on the end-to-end delay between port p_E and port $p_{E'}$. It states that the time elapsed between the passage of a frame on p_E and the passage of the frame in $p_{E'}$ must not exceed K with a probability greater or equal to β.

Notice that inequations A, B, and C express hard real-time constraints whenever β is equal to 1.

The distribution followed by a given service can be computed according to either of the following two methods:

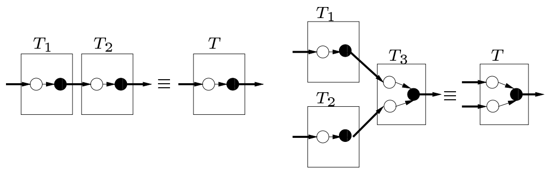
- with an approximation of the distribution by using a computation based on samples.
- by using the formal expression of the distribution, which may result from a statistical treatment of the empirical samples or from an *a priori* knowledge.

Notice that the precision is far better when the formal expression of the distribution is known and the distribution function is invertible.

4.2. Timeliness Verification Algorithm

The temporal behavior of the application is determined by applying a timeliness verification algorithm. This algorithm first composes the temporal behavior of each of the elements of the application either by a formal calculus or by simulation. Formal calculus must be preferred whenever it is possible: for example, the sum of two service times following normal distributions is equivalent to a normal distribution. When combining elements, two different kind of computations can be observed, as detailed below.

Serial Composition. Two serial elements are reducible to a single element (see Figure 7(a)): the distribution of T is given by the computation of $T_1 + T_2$.



(a) Serial Composition (b) Parallel Composition

Figure 7. Composition of Elements

Parallel Composition. Two parallel elements are reducible to a single element (see Figure 7(b)). Let $a \wedge b$ be the maximum between a and b. The distribution of T is given by the computation of $(T_1 \wedge T_2) + T_3$.

Meeting Temporal Constraints. We have to determine whether the temporal constraints expressed by inequations A, B and C can be satisfied or not. More precisely, we have to determine whether the proportion of samples that fulfill a temporal property P is greater than x or not (hypothesis H). Let α be the *a priori* risk accepted by the software architect. In other words, the software architect accepts hypothesis H although it is false with a proportion α. To evaluate H, we rely on a standard result of statistic theories [6]. Let p be the theoretical proportion of samples fulfilling the temporal property P. Let y be the actual number of samples that satisfy P, with y a binomial distribution $\mathcal{B}(n,p)$. Let $F = \frac{y}{n}$ be the empirical proportion of the samples meeting P. Then, by the central-limit theorem, F approximates a normal distribution $normal\left(p; \sqrt{\frac{p(1-p)}{n}}\right)$, with the notation $normal(m, \sigma)$ standing for a normal distribution of mean m and variance σ^2. Then if x is in the set of acceptable values, the hypothesis H can be accepted with a risk α. Otherwise, we refuse the hypothesis H. This result can be employed only if np and $n(1-p)$ are greater than 5 [6]. As an example, for a sample of size 200000 of a random variable T there are 99.999% samples meeting P. Then we can say with a confidence of 95% (that is, we accept to make a mistake in 5% of our decisions) that the percentage of samples that will satisfy P are between $0.99999 \pm \sqrt{\frac{0.99999\times(1-0.99999)}{200000}}$, which gives that 99.9976 of the samples of T must guarantee P. So if the requirements are that at least 99.995% of the samples of T are meeting P, then we may accept the hypothesis with a confidence of 95%.

Temporal Behavior of the Virtual Studio Application. Let us consider our application example. Pictures are taken approximatively with a frequency of 24Hz, *i.e.* around 42ms. Simultaneously to the pictures acquisition, position of the camera is recorded. We consider that the duration of measurements and recordings of pictures are negligible. Three temporal constraints have to be verified:

1. The background pictures computed by S_2 and taken by the camera have to reach simultaneously the mixer. Let T_E be the random variable expressing the time between the input and output of a frame in a given element E. We have:

$$\mathbf{P}[\max_{k,l\in\{1,2\}} |\delta_{i_{C_3^k}} - \delta_{i_{C_3^l}}| \leq 21] \geq \quad 1 - 10^{-4} \qquad (5)$$

This probability expresses that the difference between the time of arrival on the mixer of the background picture ($T_{\gamma_{1,2}} + T_{C_2} + T_{\gamma_{2,3}}$) and the time of arrival of the foreground picture ($T_{\gamma_{1,3}}$) should not exceed 21ms to prevent that two frames not belonging to the same scene be mixed. A probability of $1 - 10^{-4}$ means that we do not tolerate more than one mistake every 7 minutes.

2. There is less than tenth of a second between the time a picture is taken and the time it is displayed on the screen of C_5 and C_6. More formally:

$$\mathbf{P}[|\delta_{i_{C_5}} - \delta_{o_{C_1}}| \leq 21] \geq \quad 1 - 10^{-4} \qquad (6)$$
$$\mathbf{P}[|\delta_{i_{C_6}} - \delta_{o_{C_1}}| \leq 21] \geq \quad 1 - 10^{-4} \qquad (7)$$

3. The pictures have to be smoothly displayed to the user. Blanking screen must be prevented, *i.e.* there is less than one twenty fifth second between the display of two pictures. Let $\Delta(X)$ be the difference between two independent values taken by a random variable following the distribution of X, then according to the fact that $\mathbf{P}[\Delta_{i_{C_7}} \in [0,40] = \mathbf{P}[T_{C_1} + \Delta(T_{\gamma_{1,3}} \wedge (T_{\gamma_{1,2}} + T_{C_2} + T_{\gamma_{2,3}}) + T_{C_3} + T_{\gamma_{3,4}} + T_{C_4} + T_{\gamma_{4,7}} + T_{C_7}) \leq 40]$, we get:

$$\mathbf{P}[\Delta_{i_{C_7}} \in [0,40]] \geq 1 - 10^{-4} \qquad (8)$$

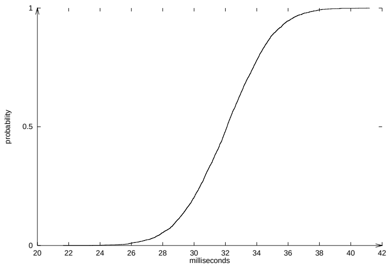


Figure 8. Temporal Constraint 5

In order to simplify the example, we are neglecting the processor loads. We suppose there is no share of CPU resources and that computers are powerful enough to complete their task. By taking some empirical values of time consumption for the treatments, it is possible to verify if the temporal constraints from (5) to (8) are met. By composing the statistical samples, we obtain curves similar to Figure 8. These curves are the characteristic functions of the distributions followed by the expressions of the constraints. Notice that by assuming that samples follow normal distributions, the distribution followed by the synchronization constraint of the mixer could be more precisely computed.

In our example, constraint (5) related to the synchronization on the arrival of pictures is not fulfilled: most of the frames of the background pictures stream and of the foreground pictures stream arrive on the mixer with a delay of more than 21ms. This comes from the absence of an element delaying the outputs of the camera. By relying on our proposal, checking the temporal behavior of the application when a new element (*e.g.* a delayer) is added between the camera and the mixer is relatively easy. We describe the temporal behavior of the application hierarchically to reduce the number of components. That is, we group together elements C_1, C_2, $\gamma_{1,2}$, $\gamma_{1,3}$ and $\gamma_{2,3}$ in element E_1 and elements C_3, C_4, $\gamma_{3,\{5,6\}}$, $\gamma_{3,4}$, $\gamma_{4,7}$ in element E_2 (Figure 9), according to both protocols and temporal behavior.

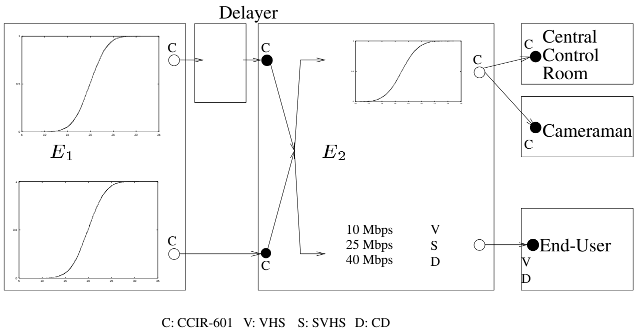


Figure 9. Temporal Grouping

Up to this point, we have provided a solution to compute a set of application states compatible with the requirements of the software architect. One of the maximal elements of this set according to the ordering relationship $\prec_{QoS}$ is chosen to configure the application at run-time. The maximal state of the virtual studio application is given by the equations: $s_E = C_E$ with $E = \{C_1, C_2, C_3, C_5, C_6, \gamma_{1,2}, \gamma_{1,3}, \gamma_{2,3}, \gamma_{3,\{5,6\}}, \gamma_{3,4}\}$, $s_{C_4} = \{((\texttt{CCIR-601}), (\texttt{any.MPEG-2.CD}))\}$, $s_{\gamma_{4,7}} = \{((\texttt{CD}), (\texttt{CD}))\}$ and $s_{C_7} = C_{C_7}$

5. Conclusion

We have presented a method to verify the correctness of multimedia applications or according to the protocols used by the elements and timeliness of the application. This method is useful when developing software out of a set of reusable elements: we can express the protocols employed at the inputs and outputs of an element, the duration elapsed between the income of a data and its output. Two algorithms have been provided to check both compatibility of protocols employed by the elements and the correctness of the temporal behavior of the composition of the elements. Furthermore, our method allows a hierarchical description of an application, which permits its progressive development. In other words, new elements can be integrated into an existing group of elements, without having to compute again the whole protocol or temporal behavior of the application: we just have to compose the result of the algorithms for the group and the new element. Thus, we have provided a method which can be conveniently exploited in an environment aimed at developing multimedia applications. This method alleviates the work of the software architect by allowing him to verify *a priori* (*i.e.* without building the application) the feasibility of its application.

Further interesting question is the management of resources: so far, we have addressed this issue only by considering that resources were reserved for the duration of the application. We could extend the management of resources to allow dynamic reconfiguration of the states taken by the applications. Thus, we have to ensure the availability of software permitting dynamic reconfigurations (*e.g.* QoS Architectures); and we have also to provide methods to define priority among streams in order to determine which ones can be lowered or improved.

References

[1] G. Blair, L. Blair, H. Bowman, and A. Chetwynd. *Formal Specification of Distributed Multimedia Systems*. UCL Press, 1998.

[2] A. T. Campbell. *A Quality of Service Architecture*. PhD thesis, Lancaster University, Jan. 1996.

[3] S. Gibbs, C. Arapis, C. Breiteneder, V. Lalioti, S. Mostafawy, and J. Speier. Virtual Studios: An Overview. *IEEE Multimedia*, Jan. 1998.

[4] N. Medvidovic and R. Taylor. A Framework for Classifying and Comparing Architecture Description Languages. In M. Jazayeri and H. Schauer, editors, *Proc. 6th European Software Eng. Conf. (ESEC 97)*, volume 1301 of *Lecture Notes in Computer Science*, pages 60–76. Springer-Verlag, Berlin, 1997.

[5] K. Nahrstedt. *An Architecture for End-to-End Quality of Service Provision and Its Experimental Validation*. PhD thesis, University of Pennsylvania, 1995.

[6] G. Saporta. *Probabilités, analyse des données et statistique*. Éditions Technip, 1996.

[7] M. Shaw and D. Garlan. *Software Architecture : Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

[8] B. Spitznagel and D. Garlan. Architecture-Based Performance Analysis. Technical report, Carnegie Mellon University, Sept. 1997.

[9] R. Steinmetz and K. Nahrstedt. *Multimedia : Computing, Communications and Applications*. Prentice Hall, 1995.