MediaWare: On Distributed Multimedia Temporal Relations and Synchronization Agents

Yahya Y. Al-Salqan†and Carl K. Chang‡

†Concurrent Engineering Research Center
West Virginia University
Morgantown, WV 26506
E-mail: alsalqan@cerc.wvu.edu

‡EECS Department (M/C 154)
University of Illinois at Chicago
Chicago, IL 60607

Abstract

A key characteristic of multimedia data such as video data is the spatial/temporal semantics associated with it. This makes the multimedia data and application more complex to generate and control. Temporal relationships are inevitable among distributed multimedia objects. Synchronization of multiple data streams in time has thus been recognized as a crucial requirement for multimedia applications. Intelligent agents has been used in telecommunication, power system control, medical applications, and intelligent document retrieval. In this paper, we propose the usage of the intelligent agents in multimedia integration. We present synchronization agents as "smart" distributed objects that deal with scheduling, integrating, and synchronizing distributed multimedia streams. The synchronization agent can handle accurate and uncertain temporal requirements which may exist in the teleorchestra-

tion of fuzzy scenarios or query. We show how the fuzzy scenarios that contain imprecise synchronization constraints, such as relative object presentation starting or ending points, can be transformed into more accurate temporal specifications. In addition, the paper includes a formal specification model, the Interoperable Petri Nets (IPNs) to describe the agents behavior and to capture the temporal semantics existing among distributed multimedia objects. The implementation of the synchronization agents is based on the IPN specification model. The paper provides several examples to show our approach.

Keywords: Agents, fuzzy scenario, multimedia integration, interoperable Petri nets (IPNs), synchronization, teleorchestration, temporal relations.

1 Introduction

Distributed multimedia applications are growing rapidly in a variety of fields, including education, entertainment, weather forecasting, and medical diagnosis. Multimedia objects include images, voice, full motion video, text and numeric data. Due to the diverse and distributed nature of multimedia objects, an application object can be composed of different but related components. For real-time playout of such inter-related streams, a synchronization technique is essential. Multimedia composition can be specified by a scenario representing the temporal and spatial relationships among the involved media objects. In [LKG94], two major problems involved in the media composition were investigated, namely the specification and modeling of the required temporal relationships and the implementation of temporal specifications by scheduling the objects involved.

The composition of multimedia teleorchestration (remote multimedia presentation) may involve imprecise and relative temporal relationships among media objects; therefore, a transformation technique to convert such imprecise scenarios into more accurate ones is needed. In this paper, we present a new approach based on synchronization agents which impose the synchronization requirements among distributed multimedia streams. Our approach includes a modeling mechanism to specify the distributed multimedia synchronization and temporal relationships among multimedia objects. Our method can deal with both accurate and fuzzy scenarios.

Synchronization, in general, can be classified into two types: intra-stream and inter-stream. Intra-stream synchronization is mainly concerned with delivering every media object in time to meet the intended playout deadline. Intra-stream deals with the low-level synchronization of an individual data object originated by any media source. Inter-stream synchronization, however, maintains the temporal relationships between different but related media streams, such as the *lip-synch* between video and audio objects[1].

The concept of an agent has become important in artificial intelligence (AI) and mainstream computer science. Distributed AI researchers have applied agent technology in a variety of areas. Example applications include telecommunication network management, concurrent engineering, power system management, air-traffic control, intelligent document retrieval, and patient care [WJ94, WJ95].

In this article, we propose a new approach that uses intelligent agents to help solve synchronization and scheduling among distributed multimedia objects. This article also provides a specification approach to model the synchronization agents. Our specification model is based on interoperable Petri nets (IPNs).

Petri Nets have been used for modeling and simulating concurrent systems since their introduction by C. A. Petri in the 1960s [Mur89, GL81, Jen81, Rei83, BGV91, Jen91]. A Petri net is a bipartite, directed graph with transitions and places. Figure 2, for example, shows a simple Petri net. In order for a transition to fire, its input places must have tokens. Transition t_1 in FigureřefOCPN is enabled and can fire since there is a token in its input place $P_{initial}$. Firing transition t_1 puts a token in each of its output places. In HLPNs, as an extension to Petri nets, colors can be associated with places and arcs. In order for a token to reside in a place or travel through an arc, the token's color must have the same color as the arc and the output place. For the purpose of defining a better formalization for distributed multimedia computing, concurrent object-orientation, concurrent engineering, heterogeneous systems and open system architecture, we introduced IPNs [AS94, ASCR95, ASC95]. IPNs extend the high-level Petri nets by adding a port and a port interface construct at the graphical notation level, and by introducing the interoperability notion at the operational level.

In our work, we introduced MediaWare as a distributed multimedia com-

[1]Refer to [ASCR95] for a complete discussion of inter- and intra-stream synchronization.

puting environment and formalism [ASC95]. The MediaWare architecture, as shown in Figure 1, includes seven layers: the application layer on top; the quality-of-connection layer, which includes the quality-of-service, synchronization specifications and agents, and user interaction; the interoperable virtual connection protocol (IVCP) layer; the object repository layer; the object request broker (ORB) layer, the interface definition language (IDL) mapping; and, finally, the object implementation layer.

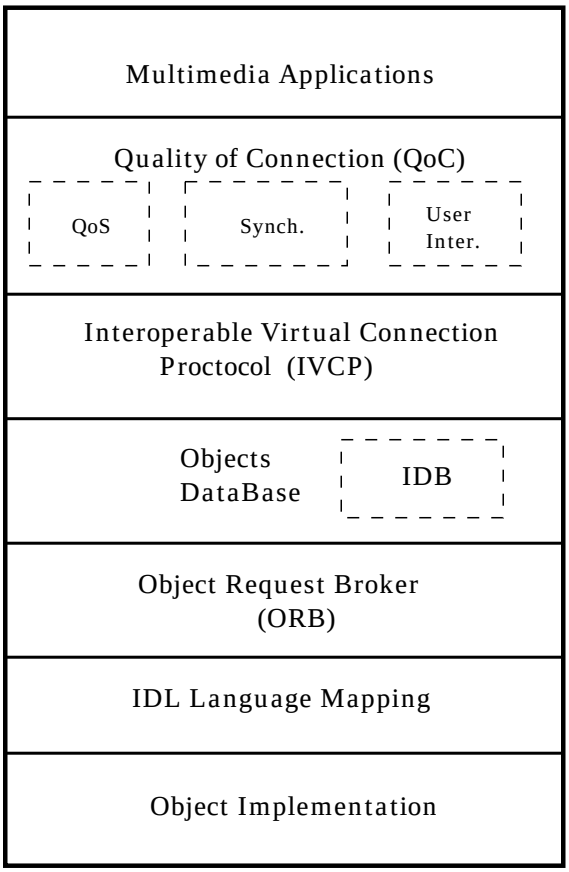


Figure 1: MediaWare's architecture.

One of MediaWare's major features is its ability to support the dynamic creation of distributed multimedia scenarios, which is the focus of this paper.

This paper is organized in the following manner: Section 2 reviews related work, Section 3 informally introduces the Interoperable Petri Nets and

discusses their firing rules[2], Section 4 presents the temporal relations existing among media objects and uses an example of the synchronization agent dealing with precise and fuzzy temporal specifications, and, finally, Section 5 offers our conclusion.

2 Related Work

The complexity of multimedia applications has grown significantly during the past several years. To tame such a complexity, AI techniques can be incorporated. In addition, formal approaches for the specification, design, and analysis of multimedia applications are highly desirable. However, few such models have been adequately defined to date. A model should be flexible and rich enough to address several multimedia issues, especially the distributed nature of multimedia resources and synchronization among their playout.

Software agents have been used and proposed in a wide range of applications [WJ94]. For example, much work on interface agents is being done by the computer-supported cooperative work (CSCW) community (see [WJ94] for details). In addition, a number of studies have been conducted on information agents. An information agent has an access to at least one and potentially many information sources. It is able to collect and manipulate information obtained from these sources in order to answer queries posed by users and other information agents. The information retrieval agent (IRA) is able to search for loosely specified articles from a range of document repositories. Another important information agent system is called Carnot [WJ94], which allows pre-existing and heterogeneous databases to work together to answer queries that are outside the scope of any of the individual databases. In this paper, we introduce the synchronization agent as a new approach to help in synchronizing and scheduling distributed multimedia streams.

Temporal relationships are inevitable among multimedia objects. In particular, the synchronization of multiple data streams in time has been recognized as a significant requirement of multimedia applications [GBC88, GC91, LG91]. Recent literature suggests several methods to address this problem at the low-level (bit-ways) [LBea90, LG91, Nic90, Ste90].

Several authors have considered formal methods for multimedia synchronization [LG91, QWG93, LG90, PR93]. In [LG91, QWG93, LG90], the Ob-

[2]For more details and a formal definition of IPNs, refer to [AS94].

ject Composition Petri Net (OCPN) and its extension (XOCPN) are discussed. An example of an OCPN specification for multimedia data is shown in Figure 2, which depicts a video clip, an audio clip, and some text data ordered in time as illustrated. Firing the transition t_1 puts a token into the 'Video' and 'Audio' places. Each of these places is assigned a duration τ, which specifies the concurrent playout of the video and audio objects for τ time units. After τ units have elapsed, transition t_2 fires to start the text object playout.

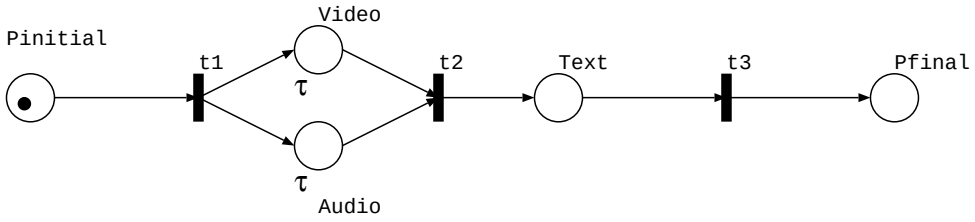


Figure 2: An OCPN example specification of playing audio and video clips followed by text.

Figure 2 can be translated into the XOCPN as shown in Figure 3. In

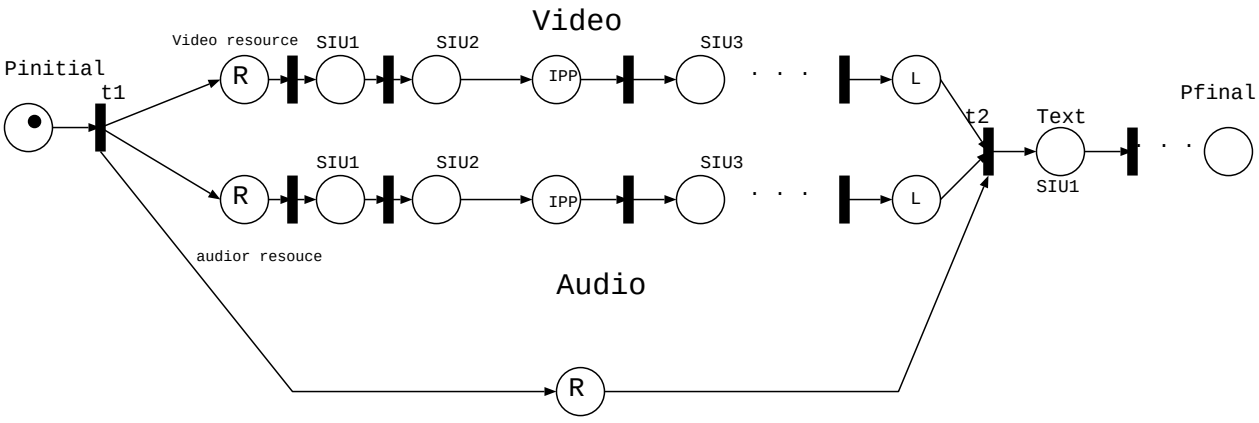


Figure 3: A XOCPN transformation from OCPN.

Figure 3, places R (Resource Setup), L (Resource Release), and IPP (Interstream Pacing Point) represent control objects and the SIU places represent Synchronization Interval Units. The playout duration of video and audio

objects, since they are synchronous, are divided equally into $SIUs$. $SIUs$ are used for intra-stream synchronization, whereas $IPPs$ are used for inter-stream synchronization.

The XOCPN (and its predecessor OCPN) does not capture the distributed nature of multimedia applications and devices. For instance, in order to allocate (release) a resource, all that XOCPN does is introduce an $R(L)$ place. Resource allocation in distributed multimedia applications is much more complex, however.

An additional disadvantage of the XOCPN model is that the resultant playouts of the synchronized streams are dependent upon each other. This does not correspond to the real-world situation. For example, a video camera captures a video stream and a microphone captures a voice stream; although the resultant playout should be synchronized, the camera and the microphone devices are completely independent and they capture and produce data concurrently. Our IPN model will address this problem satisfactorily.

Li et al [LKG94], introduced the time-flow graph (TFG) to represent the temporal relationships among media objects. For example, the TFG representation of "X overlaps Y" relation is shown in Figure 4. The square

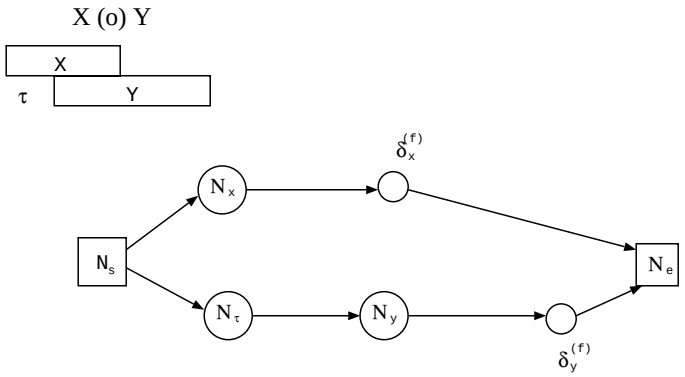


Figure 4: The time flow graph presentation of "X overlaps Y".

nodes, N_s and N_e, in Figure 4 represent a parallel start and end, respectively. The circle node, N_x, represents the playout of object X for x duration. The same is true for Y and τ objects (τ represents an intermission object). The $\delta_y^{(f)}$ and $\delta_x^{(f)}$ nodes represent the need to specify the finishing point of the x and y intervals.

Based on the above example, note the following disadvantages of time flow graphs:

- The graphs fail to represent the distributed nature of multimedia objects.
- The square nodes, N_s and N_e, have no temporal semantics. For example, when objects X and Y finish their playouts, N_e has no temporal specification to correspond to this ending. It is not considered a transition, for example.
- The introduction of δ intervals does not enforce the relationships between the related objects. For example, if both $\delta_x^{(f)}$ and $\delta_y^{(f)}$ are equal to "zero"; therefore, the relation in Figure 4 becomes "X ends Y" instead of "X overlaps Y".

In order to overcome the limitations of the XOCPN and TFG models, we introduce the IPNs model to represent the temporal relations. The IPN model preserves the interoperability characteristics of multimedia applications. Generally speaking, two or more objects/agents/applications are interoperable if they can cooperate, regardless of their underlying technology, to accomplish a joint task[3]. In [ASCR95], we used the IPNs to model the intra-stream and inter-stream synchronization requirements. In this paper, we use the IPNs model in the formal specifications and analysis of distributed multimedia synchronization agents. The agents' implementation is based on the IPNs' specifications.

3 Interoperable Petri Nets

We use interoperable Petri nets as a formal model for distributed computational agents. This model, the IPNs, extends High-Level Petri Nets (HLPNs) model by introducing port and port interface constructs at the graphical notation level and by introducing the interoperability feature at the operational level. Interoperability is achieved through the Interoperable Virtual Connection Protocol (IVCP)[4]. Our paper reveals the power of IPNs to model synchronization among distributed multimedia objects.

[3]This is our own definition of interoperability.

[4]Refer to [AS94] for a full description of IVCP.

An IPN model, informally, consists of a processing element, ports, and ports' interfaces. The processing element is represented by a HLPN; ports can be input or output kind; and ports' interfaces are the visible part of the IPN module where data can be consumed or produced. In the following, we introduce the necessary definitions to describe the IPN more formally.

Definition 3.1 (IPN Module) *An abstraction and encapsulation of processing element(s), port(s) and port interface(s). A module can encapsulate more than one port and/or more than one processing element.*

(In this paper, we use the term IPN module and IPN object interchangeably.)

Definition 3.2 (IPN Processing Element) *The internal decomposition of the IPN module which describes the module behavior in response to external or internal stimuli. The processing element defines the behavioral semantics of the IPN module. Processing elements are represented by HLPNs.*

Definition 3.3 (IPN Port) *The component of the module (could be composite) which produces or consumes messages*[5]*. Each port on an IPN is associated with a specialized interface.*

An input port consumes messages coming from other modules, and an output port produces messages to be sent to other modules.

Definition 3.4 (IPN Port Interface) *A set of valid message types which are allowed to pass through the port. A port interface is the opening of the encapsulation shell of an IPN object (module) and the visible part to other modules.*

Definition 3.5 (Messages) *Discrete units of information representing the sole means of interaction between IPN modules. Messages, represented by colored tokens, are sent and received through ports and travel through channels.*

Definition 3.6 (IPN Virtual Connection) *Two or more IPNs are virtually connected by establishing a communication channel between their input and output ports. The virtual channel is established by applying the interoperable virtual connection protocol*[6]*. The ports involved must have compatible interfaces.*

[5] The definition of IPN port is different from the definition in [Jen91].

[6] Refer to [AS94] for a full description of IVCP.

Definition 3.7 (Virtual IPN) *The IPN resulting from combining multiple IPN modules to accomplish a certain task. A virtual IPN has a temporal behavior allowing the connection among the constituent IPNs to be disconnected after the task has been accomplished.*

In order for two (or more) IPNs to interoperate, a virtual connection must be established first between the source's output port interface and the destination's input port interface. There are two different types of possible virtual connections associated with two different types of ports:

- A connection may pass messages "colored tokens" from a transition to a place. In this case, the transition is the output port in the source IPN module and the place is the input port of the target IPN module.
- A connection may pass color tokens from a place to a transition. In this case, the place is the output port in the source IPN module, whereas the transition is the input port in the target IPN module.

In IPNs, the distinction between these two types of connections is made clear by the corresponding interfaces' icons[7]. We introduce IPN icons for port interfaces in Figure 5. IPN icons include place and transition seekers in addition to place and transition receptors.

A place seeker is used as an interface for a transition output port, while a place receptor is used as an interface for a place input port. A transition seeker is used as an interface for a place output port, while a transition receptor is used as an interface for a transition input port. In the following, we exemplify a general description of IPNs.

The IPN net in Figure 6 is a simple representation of a microphone and a speaker devices. In the initial marking, there is one colored token, $< T = audio, BW = 100, D = mic >$, at place A. The IPN in Figure 6 consists of four parts:

- The net structure.
- The declarations.
- The net inscription.

[7]Some of them are similar to the icons in [Neu93].

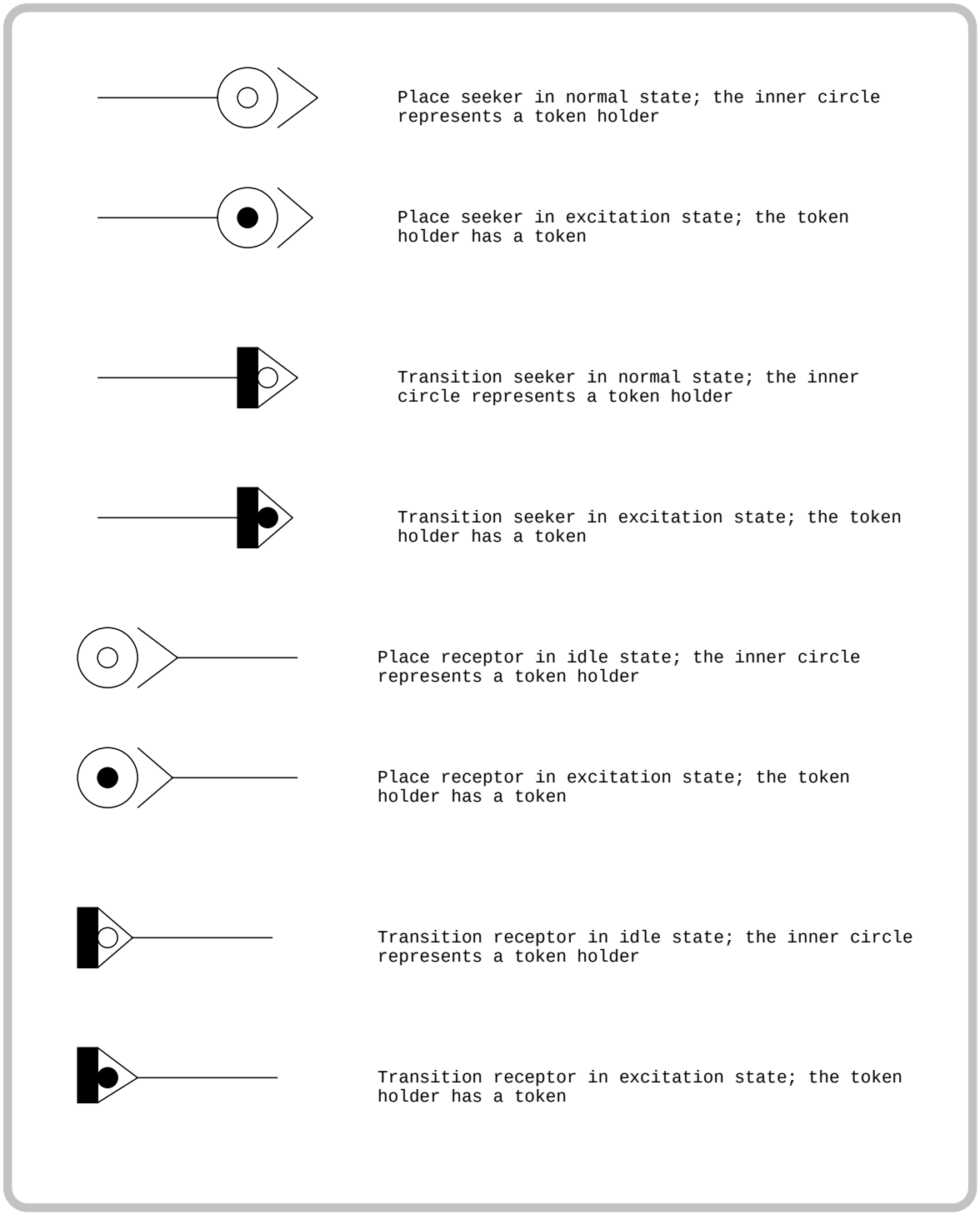
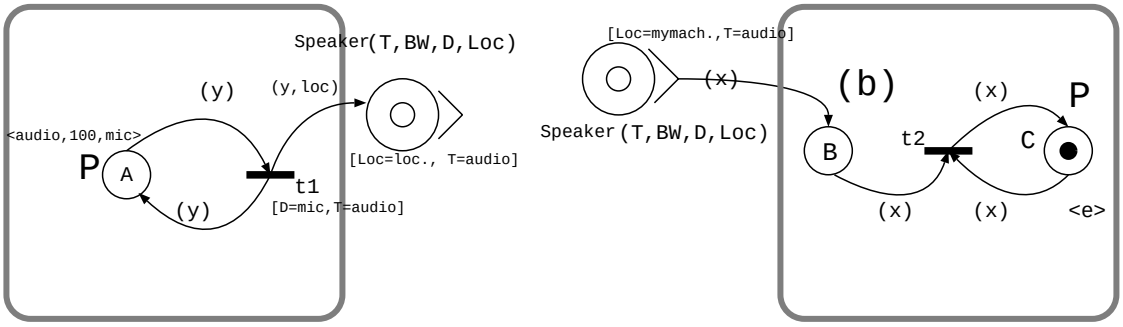


Figure 5: IPN icons.




```
Color TType T={audio}
Color BWType BW =[80 ... 100]
Color DType D = {mic, file}
Color EType E ={e}
Color LocType Loc= {machine addresses}
Color PType P = Product TType * BWType
                       * DType
Var x, y = P
```

(a) Microphone

```
Color TType T={audio}
Color BWType BW =[70 ... 120]
Color DType D = {speaker, mic, file}
Color EType E ={e}
Color LocType Loc= {machine addresses}
Color PType P = Product TType * BWType
                       * DType
Var x, y = P
```

(b) Speaker

Figure 6: Two IPN modules, a Microphone (a) and a Speaker (b).

- The port interfaces.

The net structure is a bipartite, directed graph with two kinds of nodes: places and transitions. In Figure 6, the net structures, enclosed in the shaded boxes, represent the IPNs' processing elements. Specifically, Figure 6(a) represents the microphone behavior specification, and (b) represents the speaker counterpart.

The declarations specify the types of color sets and variables. They are analogous to type declarations in programming languages [Jen91]. Each place has a color type associated with it, and a token must be of the same type (color set) as the place in order to reside in it. In this example, *color TType* T is a set of device type (audio). *Color PType* P is a product of color types *TType*, *BWType* (bandwidth of range type), and *DType* (enumerating device kind). Finally, we present an uncolored token by e of type *EType*. Therefore, in order for a token to reside in place A, it must have a token with a color value of $< audio, 80 \leq BW \geq 100, mic|file >$.

The net inscriptions are attached to places, transitions, arcs, port interfaces, or IPN modules. In Figure 6, places, such as A, have three different

types of inscription: names (e.g., A), color sets (e.g., P), and initialization expression (e.g., $< TType = audio, BW = 100, DType = mic >$). Transitions, such as t_1, have two kinds of inscriptions: names (e.g., t_1) and guards (e.g., $[D = mic, T = audio]$). Unlike a place name, the port interface name has a semantic use. We can regard it as a function, a procedure, or a method name. The IPN module has one kind of inscription, its name (e.g., Microphone).

Each port has a corresponding port interface (e.g., place seeker attached to transition t_1). A transition (place) output port has a place (transition) seeker as a port interface. On the other hand, a transition (place) input port has a transition (place) provider as a port interface. A guard is a boolean expression[8] that must be evaluated to true in order for a transition to fire or a port interface to be promoted into an excitation state. For example, the component D in the variable y must be equal to mic and T must be equal to $audio$ in order for transition t_1 to fire. As a convention, guards which always evaluate to true are omitted from the net. The arc label may contain variables (such as Y), constants (such as loc), and functions and operations that are defined in the declarations.

The IPN description reflects the autonomous behavior as an important intelligent agent's property.

3.1 IPN's Firing Rules

Before any transition can be enabled in an IPN (similar to HLPNs), all variables in the arc label function must be bound to colors of the corresponding types. According to this binding, if the guard is evaluated to true, the transition fires thus removing tokens from its input place(s) and adding tokens to its output place(s). The number of removed and added tokens is determined by the value of the corresponding arc label function.

If a transition is an output port of an IPN module (as shown in Figure 6(a)), its firing produces colored tokens which are added to its port interface. Consequently, if the interface is at its normal state, it will be promoted into the excitation state.

If a place is an output port of an IPN module, after receiving a token, the arc label function of the arc connects the port to its interface (transition seeker) which will then be evaluated and bound to the colors of the

[8]Enclosed in square brackets ([ ]).

corresponding types. According to the binding, the port interface will be promoted into the excitation state only if the guard is evaluated to true.

Since we use IPNs to model multimedia synchronization and synchronization deals with the temporal behavior of related streams, we include the notion of time into IPNs (with the same conventions of timed Petri nets). With timed IPNs, we are free to choose the granularity of synchronization by assigning time durations to places.

The firing rules of a timed IPN are similar to the rules of timed Petri nets:

1. A transition, t_i, fires immediately when each of its input places connected through regular arcs contains an unlocked number of tokens matching the corresponding arc labels, and these tokens evaluate the transition's guard to true. If the transition is an output port, its interface guard must also evaluate to true before the transition fires. If t_i is incident on an inhibitor arc[9], the place incident on that arc must be empty in order for t_i to fire.

2. Upon firing, the transition, t_i, removes tokens from each of its input places and adds tokens to each of its output places according to arc labels.

3. After receiving a token, a place, p_i, remains in the active state for the interval assigned to it which is specified by a duration, T_j. During T_j, the token is locked. When a place, p_i, becomes inactive (T_j expires), the token becomes unlocked.

We use the IPN model to formalize distributed multimedia applications and environments. Henceforth, we refer to the distributed multimedia environment and its formalism as *MediaWare* [ASC95]. MediaWare includes multimedia synchronization, multimedia quality-of-service, and user interactions. This article is devoted to distributed multimedia synchronization agents and their capabilities to schedule fuzzy and accurate scenarios based on temporal relations among media objects.

[9] An arc with a circle instead of arrow in its end.

4 Temporal Modeling of Multimedia Objects

Synchronizing several distributed and independent media sources is a challenging task especially if it includes uncertain temporal requirements. In teleorchestra applications (remote multimedia presentation), fuzzy presentation scenarios may be involved. In this section, we refer to multimedia integration as scenarios. Multimedia scenarios represent the temporal and spatial relationships among different data objects [LKG94]. We will discuss the temporal relationships between media objects in depth throughout this section.

The initiator of multimedia orchestrations or multimedia query may not, and should not, have the temporal specifications among all the participating objects before the starting point of the teleorchestra. Despite the lack of this inter-related temporal information, very complex temporal relationships can be required among media objects [LKG94]. For example, a user may specify a fuzzy scenario, such as object X needs to be played *during* the playout of object Y. This scenario does not determine the length of the playout of the object X. It does not determine precisely either the starting or the ending point of object X. One can infer from this scenario that object Y has a longer lifetime than X, and Y starts before X and ends after X. In our model, it is the synchronization agent's responsibility to map the fuzzy scenario into more accurate ones.

Steinmetz in [Ste92] discusses the relative synchronization operations which include imprecise and relative temporal requirements among the participating objects. In contrast to the absolute synchronization operation, the relative synchronization operation is described only by certain references without explicit time parameters, such as the starting and ending points. Nevertheless, the relative synchronization requirements have to be transformed into absolute synchronization requirements [Ste92]. In [WWR95], the $TIEMPO$ environment is described through which the authors model the effect of users' interactions on the temporal layout of multimedia titles. $TIEMPO$ discusses the need for temporal interactions which affect the final presentation schedule at run time. $TIEMPO$ assumes accurate temporal layout. In this paper, in addition to the user's interaction influence on the multimedia layout, we also consider the case where the schedule itself could be fuzzy. We provide a specification model to support the transformation from fuzzy into accurate specifications. Our model is based on the temporal-

interval relations and IPNs.

4.1 Temporal Intervals in Multimedia Teleorchestra and Query

Time can be specified in two ways: as a point in time or as an interval. Multimedia objects always have durations in their playout. As a result, the notion of temporal interval is taken as the primitive in our model.

A temporal interval is a non-zero duration of time whereas, a time instant is a zero-length time duration (e.g., "9:00 AM"). A simple logic of temporal interval is introduced in [All83].

Figure 7 gives an example of temporal relationships existing among data objects in a teleorchestra. In this example, several distributed media objects are involved in a scenario. The scenario starts with a picture $P1$. When $P1$ finishes, picture $P2$, text $T1$, and audio $A1$ objects start simultaneously. After finishing $P2$, a video clip $V1$ begins. The playout of $V1$ overlaps with that of $T1$, and $V1$ occurs during $A1$ playout. When $A1$ finishes, another video clip ,$V2$, starts which is followed by the audio object $A2$.

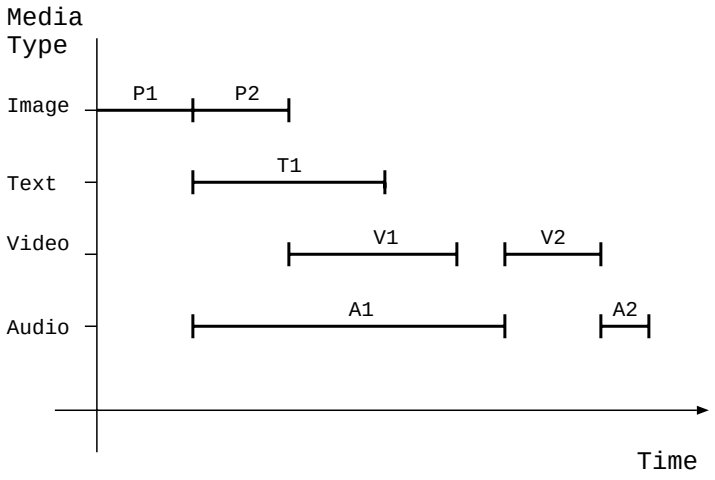


Figure 7: An example of temporal relations in multimedia scenario.

According to Allen [All83], there are thirteen different temporal relationships (seven original with their associated inverses). These relationships, as

shown in Figure 8, are *before, meet, equal, overlap, during, start, and finish* with their corresponding inverses.

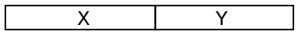
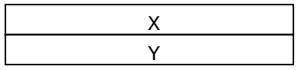
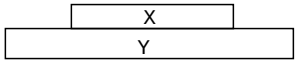
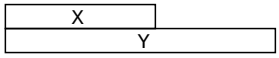
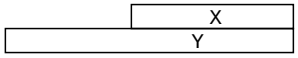
Relation	Pictorial Example	Symbol	Inverse
X Before Y		X (b) Y	X (bi) Y
X Meets Y		X (m) Y	X (mi) Y
X Equals Y		X (e) Y	X (ei) Y
X Overlaps Y		X (o) Y	X (oi) Y
X During Y		X (d) Y	X (di) Y
X Starts Y		X (s) Y	X (si) Y
X Finishes Y		X (f) y	X (fi) y

Figure 8: The temporal relationships.

Based on Figure 8, the scenario in Figure 7 can be modeled by the temporal chart shown in Figure 9.

Now we can represent the temporal chart specified in Figure 9 by the following temporal relation:

$[\{(P1)(m)\{(P2)(s)\{(T1)(s)(A1)\}\}\} \wedge \{(T1)(o)(V1)\} \wedge \{(V1)(d)(A1)\}$
$\wedge \{(A1)(m)\{(V2)(m)(A2)\}\}]$.

Multimedia teleorchestration needs to be flexible in order to allow the designer to choose the time frame (interval) of the playout of the teleorchestra playout. For example, a user could specify (choose) that one needs the teleorchestra to be confined to the length of certain object(s). MediaWare supports the choose function as a meta-relation between related objects. The

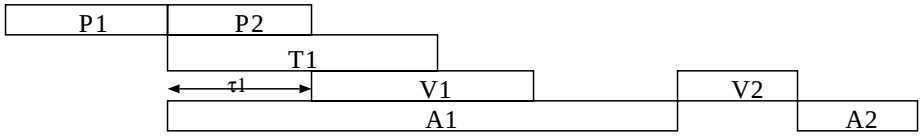


Figure 9: The Temporal Chart for the Teleorchestra.

following example illustrates the specification of the *choose* function:

Example with the choose function: Assume that a multimedia teleorchestra is composed according to the temporal relationships as specified in Figure 10. The teleorchestra includes the following objects: a text object (T1), two video objects (V1 and V2), and an audio object (A1). The text object (T1) starts the scenario. The playout of the text object (T1) includes the playouts of the video objects (V1 and V2) , and overlaps with the playout of the audio object (A1).

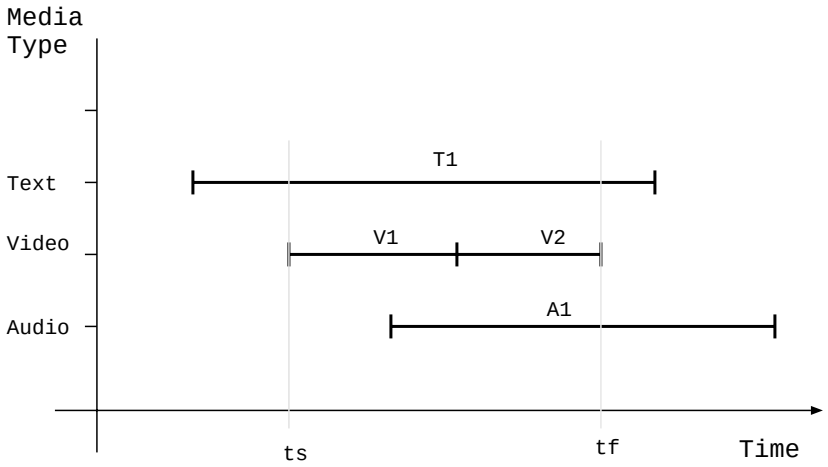


Figure 10: A Multimedia Teleorchestra with the Choose Function.

The above temporal relations can be expressed as:

$[\{(V1)(m)(V2)\}(d)\{(T1)\} \wedge \{(T1)(o)(A1)\}]$.

Now assume that a certain user requires the playout of these objects (T1, V1, V2, and A1) be confined to the playout time of the two video

objects (V1 and V2). Therefore, the playout of the text object 'T1' before 't_s' as well as the playouts of 'T1' and 'A1' after 't_f' will be skipped. As such, we can transform the above choose relationship (ch) as follows:

$ch[\{(V1)(m)(V2)\}(d)\{(T1)\}\wedge\{(T1)(o)(A1)\}] = [\{(V1)(m)(V2)\}(e)\{(T1)\}\wedge \{(T1)(f)(A1)\}]$.

Note that the interval (T1) in {(T1) (f) (A1)} is the new interval (T1), which is equal to the interval {(V1) (m) (V2)}.

In general, the choose function can be replaced equivalently with other temporal functions according to the following theorem.

Theorem 4.1 *Let the interval 'X' be associated with the interval 'Y' with a relation (r), i.e. X (r) Y, and the choose function specified as* $ch(X(r)Y) = X$. *Then, the choose function can be replaced according to the following*[10]*:*

1. $ch(X(d)Y) = X \Longrightarrow X(e)Y$,
2. $ch(X(f)Y) = X \Longrightarrow X\ (e)\ Y$,
3. $ch(X(m)Y) = X \Longrightarrow X$,
4. $ch(X(b)Y) = X \Longrightarrow X$,
5. $ch(X(e)Y) = X \Longrightarrow X\ (e)\ Y$, *and*
6. $ch(X(o)Y) = X \Longrightarrow X\ (f)\ Y$.

Based on the above discussion, in the following we provide our approach to model the temporal specifications among multimedia objects.

4.2 Modeling the Temporal Intervals and Synchronization Agents

In our model, we associate a time interval with a place. When a token enters a place, it will be in the locked state within the interval associated with that place.

Before we proceed with the modeling of temporal relations, let us enumerate the following conventions:

[10]The proof is provided in Appendix A.

- The interval in IPNs, representing the playout of the media object, is rendered by producing tokens, which correspond closely to real-world situations.

- The time delay assigned to a transition is represented by δ. As a naming convention, if δ equals zero, it is not shown on the IPNs model.

- Tokens have three different types: x, y, and τ. The x (y) is the duration length of the playout of the X (Y) media object, where τ is the length of the intermission period.

- When the temporal specifications are not precisely defined, tokens (x, y, and τ) are represented by an empty token (e). For example, the meet relation (X (m) Y) does not impose any temporal specification of $X's$ starting or $Y's$ ending points. Therefore, the duration length, X, is not well defined; it can be represented by the empty token (e). A small playout interval, ϵ, is associated with the empty token.

- In the case of an empty token (e), the playout duration is not predetermined; therefore, the token needs to be returned back to its original place to guarantee the playout progress.

- The arc label function may include a condition. We represent that by equating the token variable with a specified condition value. For example, $< X >= e$ means that if the variable X equals the empty token, e, then the arc carries that token to its output place.

- Each temporal relation has a certain implication (constraint) associated with it. For example, the relation $XbeforeY$ implies that the intermission period between X and Y must be greater than zero.

- Our IPN model has a dynamic behavior. It is capable of modeling both fuzzy and precise temporal specifications. For example, if the interval X equals '5' minutes, as a precise specification, the interoperable virtual connection protocol will pass this value on to the media source object. Consequently, according to the source firing rate, ω, the source will produce $\lceil 5 * 60/\omega \rceil$ tokens. In case of the fuzzy specification, however, the synchronization agent keeps producing the empty tokens (e), each of which has a small playout period, ϵ. The empty token is then passed

to the source media object to produce one token. The flow of the empty tokens continues as long as the presentation (teleorchestra) is in progress.

Based on the above conventions and the IPNs' firing rules stated earlier, in the following we present the IPN modeling of the synchronization agents.

X before Y Figure 11 represents an IPN model of the synchronization agent for the temporal relation, $XbeforeY$ (x (b) y). This model represents the specification of the $XbeforeY$ agent's implementation.

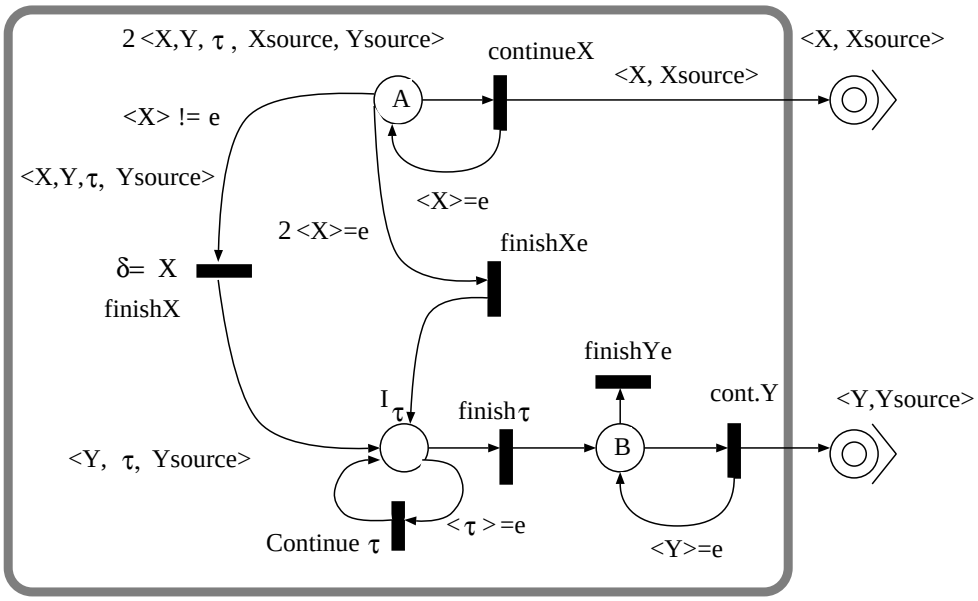


Figure 11: An IPNs model of x (b) y.

As an initial marking, two tokens of $< x, y, \tau, Xsource, Ysource >$ colors will be placed at place A. If the duration x is precisely specified (not empty), then it will be put on the output port interface (place seeker). After a delay $\delta = x$, transition $finishX$ fires and moves a token into place τ (the intermission playout). However, if x is not precisely specified (fuzzy, i.e., $x = e$), a small interval, ϵ, is assigned to the token x to be passed to the output port interface. The transition, $continueX$, keeps firing as long as the X playout is in progress. When

X playout finishes, the transition $finishXe$ fires and moves two tokens from place A into place τ.

At place τ, if the intermission period is well determined, then I_τ is specified accordingly. Otherwise, I_τ is assigned a small value, ϵ, and the transition $Continue\tau$ continues to fire as long as the intermission period is in progress. Whenever the intermission period is over, the transition $finish\tau$ fires placing a token with y value into the place B. Similar to $< x >$, the $< y >$ token is processed and placed into the output port interface (place seeker).

Placing a token on the output port interface promotes the interface from the normal state into the excitation state. This makes the output port interface ready to deliver that data to its destination specified by $Xsource$ or $Ysource$.

The modeling of the media source object X or Y is shown in Figure 12. According to the IVCP, there will be a virtual connection between

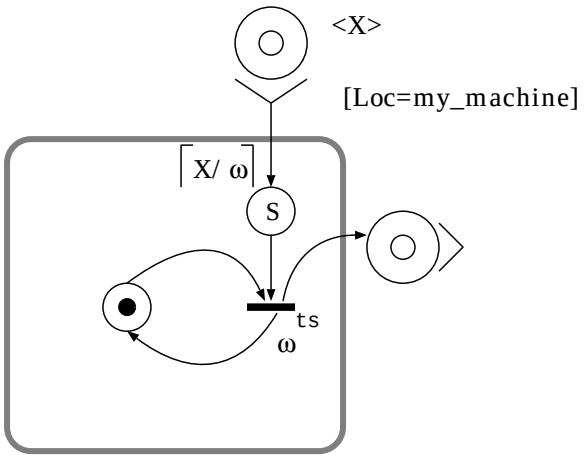


Figure 12: An IPNs model of a media source.

the $< X, Xsource >$ ($< Y, Ysource >$) output port interface in the temporal relation object shown in Figure 11, and the $< X >$ ($< Y >$) input port interface of the media object shown in Figure 12. The IVCP moves tokens from the output port interface into the input port interface. The number of tokens placed into place S is determined

according to the $\lceil x/\omega \rceil (\lceil y/\omega \rceil)$ value, where ω is the device firing rate. For example, if X is a video device, then $\omega = \frac{1}{30}$. This keeps firing the transition t_s, as long as there are tokens in place S, and produces output data (frames) for further processing at the target (display) device.

X during Y Figure 13 depicts the synchronization agent which controls the source objects, X and Y, where the temporal relation between the playout of the object X is during the playout of the object Y.

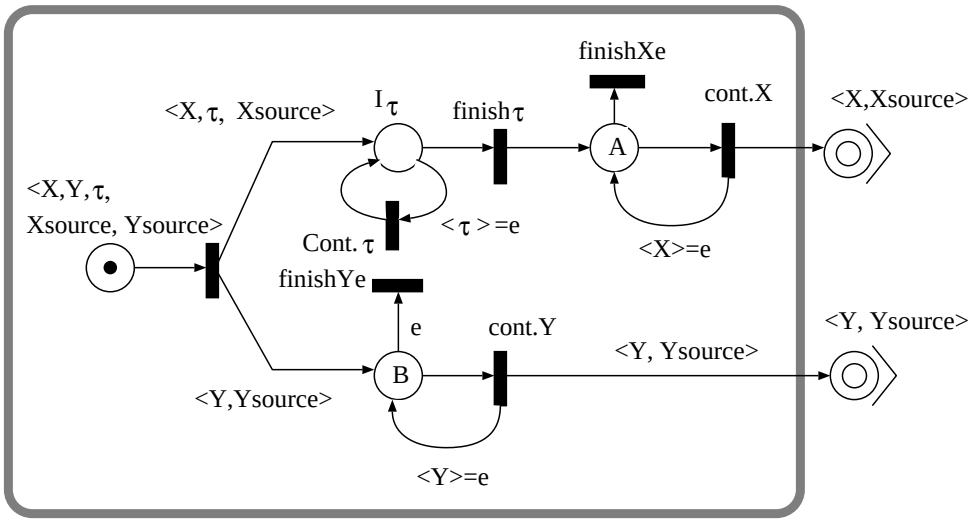


Figure 13: An IPNs model of x (d) y.

Firing the *start* transition puts a token with the color $< y, Ysource >$ which arrives at the place B and another token with the color $< x, \tau, Xsource >$ at the place τ. While the intermission period τ is in progress, B produces tokens for processing. After the intermission period expires (finishes), the A place starts producing the control tokens for the media source object X.

By the temporal implications inherited from the *during* relation, the intermission period τ should be greater than *zero* and $\tau + x$ is less than y.

The modeling of the meet, equal, overlap, start, and finish relations can be found in Appendix B.

Based on the above modeling, we present the following example to illustrate the usage of the synchronization agents. The example shows the representation of precise and fuzzy specifications of the temporal intervals.

Example 1: A client specifies the following temporal scenario for a presentation: "play a video object during the playout of an audio object, the video object starts at '10:00 AM' for '5' minutes, while the audio object starts at '9:55 AM' for '15' minutes".

This scenario is precisely specified, and the temporal relations and intervals are well identified between the audio object (X) and the video object (Y). The time chart shown in Figure 14 depicts the corresponding temporal specifications.

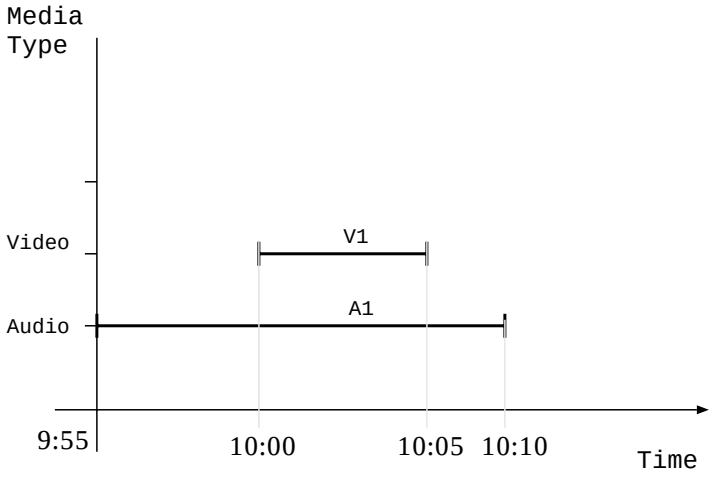


Figure 14: The time chart of the temporal relation example.

In Figure 14, the interval $Y = 15$ minutes, $X = 5$ minutes, and the intermission period $\tau_s = 5$ minutes.

In Figure 13, the starting token at the source place has the color value of <15, 5, 5, Xsource, Ysource>. When the transition *start* fires, it puts a token, $< 5, 5, Xsource >$, into place τ, and puts another token, $< 15, Ysource >$, into place B. This fires the transition $cont.Y$ and puts the token, $< 15, Ysource >$, into the output port interface. Consequently, this promotes the place seeker into the excitation state.

According to the IVCP, there will be a connection between the output port interface and the source input port interface (shown in Figure 12). Let us assume that ω is 1/20 for the Y (audio) object. Therefore, when the input port interface receives the token, $< 15 >$, it puts $(15)(20)(60) = 18000$ tokens into place S. This produces '18000' segments of audio objects for processing. Tokens represent the real data stream arrivals in real-time systems.

After the intermission period finishes, the transition $finish\tau$ fires and puts the token, $< 5 >$, into place A. This fires the transition $cont.X$, and moves the token, $< 5 >$, into the output port interface. This, in turn, promotes the place seeker into the excitation state with a token color of $< 5, Xsource >$. This token specifies the playout interval and the media source address, $Xsource$. This token is moved by the IVCP into the media source input port interface. Assume $\omega = 1/30$ for the video object (device), therefore, $(5)(30)(60) = 900$ tokens, corresponding to '900' frames, will be put into place S for processing.

In the above discussion, we explained an accurately specified temporal relation between the two objects. However, if the user does not specify the x and y durations, the starting token put into the source place has the color value of $< e, e, e, Xsource, Ysource >$. Firing the $start$ transition will move a token with a color value of $< e, Ysource >$ into place B, and a token with a color value of $< e, e, Xsource >$ into place τ. For instance, let us take the B direction (the lower part of Figure 13). The transition $cont.Y$ continues firing and producing $< e, Ysource >$ tokens and places them into the output port interface. This promotes the output port interface into the excitation state. When the virtual connection is established, according to IVCP, the stream of $< e >$ tokens continues to flow into place S at the media source object. When the user finishes the playout of the Y object, the synchronization agent stops producing empty tokens (e), and therefore, the virtual connection will be disconnected.

Example 2 This example deals partially with the scenario specified in figure7. The discussion below describes the what takes place at each point in time.

- At t=0, image 1 is started for P1 duration. The image is pro-

duced by the image source object as specified by Figure12 with $\lceil X/\omega \rceil$ equals P1. At t=0 also the x (m) y object, specified in Figure21 will be instantiated with X=P1, Xsource=the address of the image 1 object, Y=P2, and $Ysource$ will be determined later. Finally, at t=0 there will be a virtual connection between the $< X, Xsource >$ interface in Figure21 and the $< X >$ interface in Figure12.

- At t=P1, three source the image P2, the text T1, and the audio A1 objects will be launched. An x(s) y object will be started. This object, however, has three branches not two as specified in Figure24. The IVCP will group these objects in one group and that will determine the above $Ysource$ variable in the x (m) y object. According to the IVCP, there will be a virtual connection between $Ysource$ and the grouped object.

  Similar activities will take place at t = P1 + P2 since P2 meets V1, and at t = P1 + A1 since A1 meets V2, and finally at t = P1 + A1 + V2 since V2 meets A2.

In the above examples, we showed that our synchronization agent model for the temporal relations among media objects is flexible and rich enough to deal with *relative* (fuzzy) and *absolute* temporal requirements. The synchronization agents have several advantages including the transformation of fuzzy temporal specifications into more accurate ones. It helps creating more simple user interface to schedule multimedia scenarios and titles. The authors are working on the visual synchronization language which uses the synchronization agents to transform a gantt chart scenario into temporal relationship model which in turn enact the synchronization agents. Finally the synchronization agents facilitate users' interactions with multimedia scenarios and titles and help creating interactive timely-based applications.

5 Conclusion

In this paper, we have introduced the synchronization agents as "smart" objects to synchronize and schedule distributed multimedia scenarios. We showed through examples that the synchronization agent is capable of deal-

ing with absolute and fuzzy temporal specifications. It is the agent's responsibility to map the fuzzy specifications into more accurate ones.

The paper includes a description of all the temporal relations and the corresponding synchronization agents. We used the Interoperable Petri Nets (IPNs) for modeling and specifying the synchronization requirements and temporal relationships among distributed multimedia objects. The implementation of the synchronization agents is based on the IPNs' specifications.

We are using the distributed objects as specified by CORBA [Gro92] to implement the agents. CORBA provides Dynamic Invocation Interface (DII) capabilities which allows an application to issue requests for an object interface even if that interface was unknown at the compilation time. Invocations can be constructed at run time. We took advantage of this feature to build our temporal scenarios as described in the above example. CORBA, however, adds some performance degradation. In addition, The port interfaces discussed earlier are represented by the Interface Definition Language (IDL). It is our intention to produce a synchronization library which can be included in any multimedia application.

Acknowledgment

The authors would like to express their deepest gratitude to Professor Ugo Buy from the University of Illinois at Chicago for his valuable comments and suggestions. Special thanks is due to the anonymous referees for their valuable suggestions.

Appendix A: Theorem 4.1 Proof

Proof: 1. $ch(X(d)Y) = X \Longrightarrow X(e)Y$. As shown in Figure 15, the remaining playout length of the 'Y' interval after cutting its playout before 't_s' and after 't_f' is equal to the playout length of the 'X' (X (e) Y).

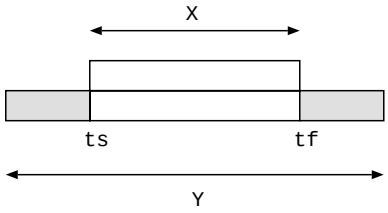


Figure 15: A Representation of $ch(X(d)Y) \Longrightarrow X(e)Y$.

2. $ch(X(f)Y) = X \Longrightarrow X(e)Y$. As shown in Figure 16, the remaining playout length of the 'Y' interval after cutting its playout before 't_s' is equal to the playout length of the 'X' (X (e) Y).

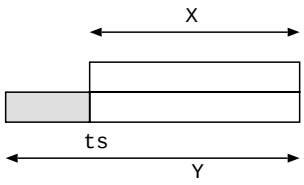


Figure 16: A Representation of $ch(X(f)Y) = X \Longrightarrow X(e)Y$.

3. $ch(X(m)Y) = X \Longrightarrow X$. As shown in Figure 17, the entire playout of the 'Y' interval needs to be cut. Therefore, the entire playout is equal to the playout length of the 'X' object (X), only.

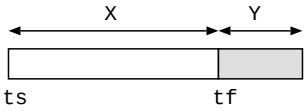


Figure 17: A Representation of $ch(X(m)Y) = X \Longrightarrow X$.

4. $ch(X(b)Y) = X \Longrightarrow X$. As shown in Figure 18, the entire playout of the 'Y' interval needs to be cut. Therefore, the entire playout is equal to the playout length of the 'X' object (X), only.

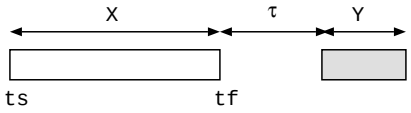


Figure 18: A Representation of $ch(X(b)Y) = X \Longrightarrow X$.

5. $ch(X(e)Y) = X \Longrightarrow X(e)Y$. As shown in Figure 19, there is no need for any transformation.

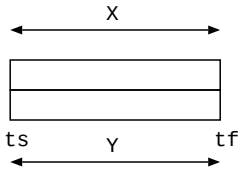


Figure 19: A Representation of $ch(X(e)Y) = X \Longrightarrow X(e)Y$.

6. $ch(X(o)Y) = X \Longrightarrow X(f)Y$. As shown in Figure 20, by cutting the playout of the 'Y' interval after the time 't_f', the remaining playout of the 'Y' interval finishes at the same time as 'X' (similar to Figure 16). Q.E.D.

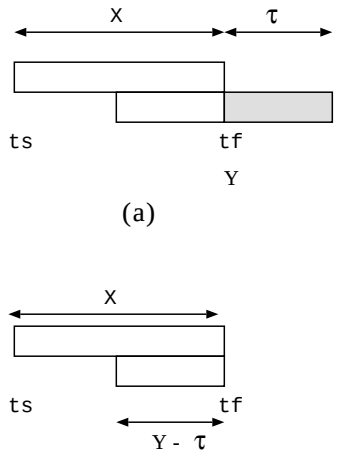


Figure 20: A Representation of $ch(X(o)Y) = X \Longrightarrow X(f)Y$.

Appendix B: Modeling of Some Temporal Relations

X meets Y X meets y (x (m) y) is a special case of 'x before y' relation, where the intermission period, τ, equals zero.

Figure 21 models the relation x (m) y.

The same discussion of 'x before y' applies here.

X equals Y Figure 22, below, depicts the IPN modeling of the 'x equals y', x (e) y, synchronization agent.

Since x equals y, only one token with the $< x, y, Xsource, Ysource >$ color is placed into the place shown in the figure. If x is precisely specified (not fuzzy), then the transition $continueX$ fires and puts a token in both output port interfaces (place seekers). Similar to the previous discussion, virtual connections take place between these output and input port interfaces of the corresponding media sources.

X overlaps Y Figure 23, shown below, depicts the IPN modeling of the 'x overlaps y', x (o) y, synchronization agent.

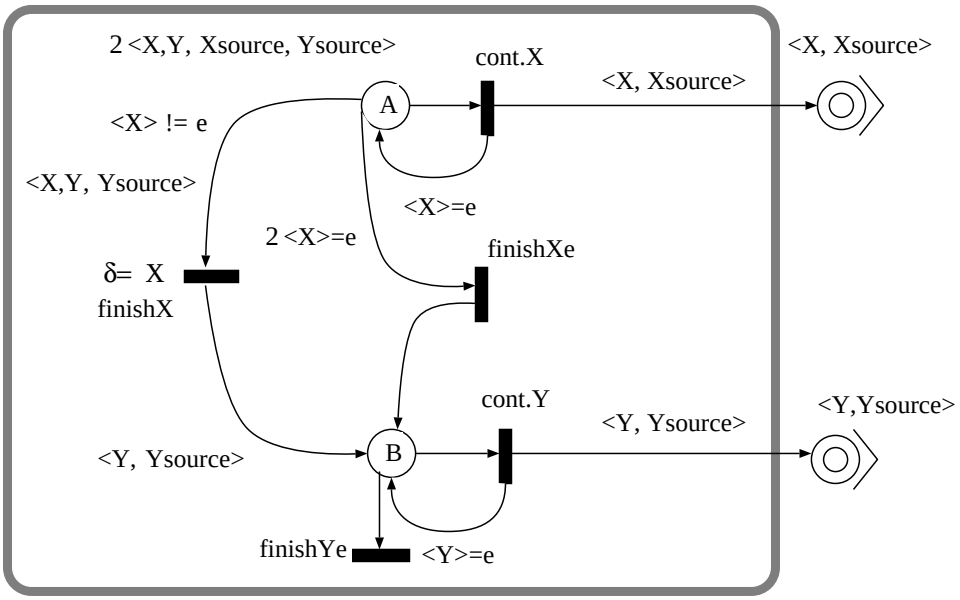


Figure 21: An IPNs model of x (m) y.

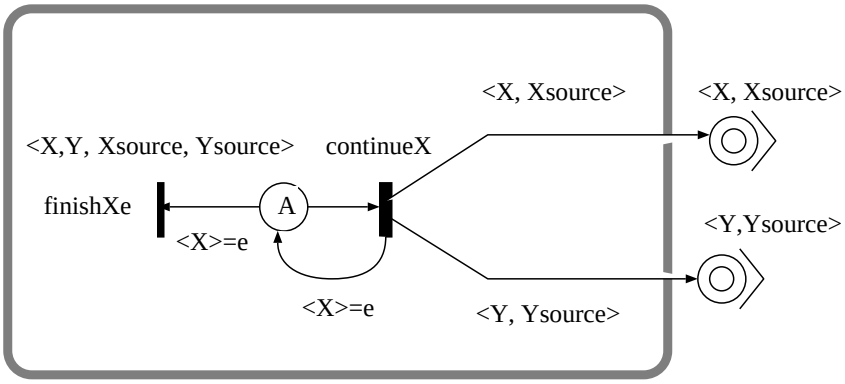


Figure 22: An IPNs model of x (e) y.

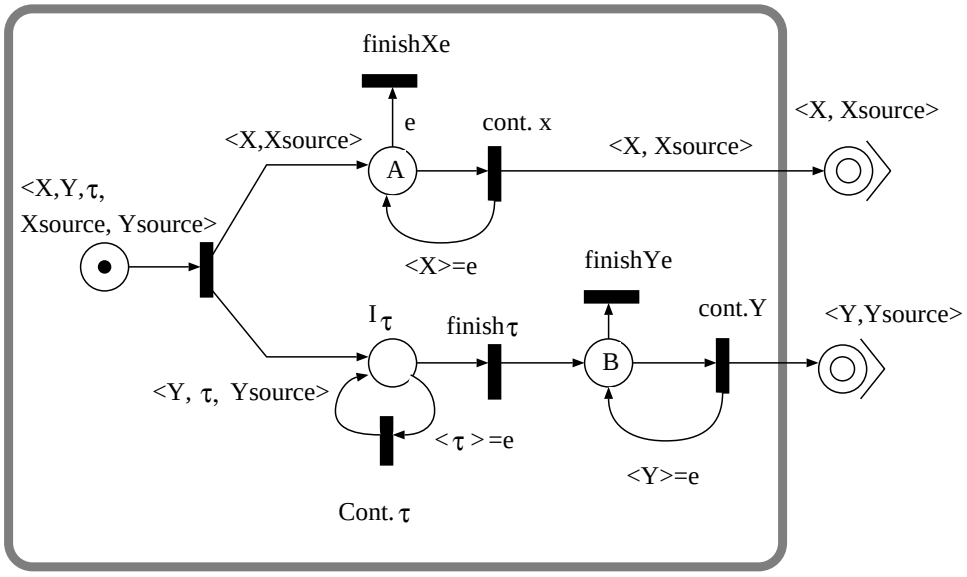


Figure 23: An IPNs model of x (o) y.

When a token with the color $< x, Xsource >$ arrives at the place A, a token with the color $< y, \tau, Ysource >$ arrives at the τ place. While the intermission period is in progress, A produces tokens for processing. After the intermission period expires (finishes), the B place starts producing the control tokens for the media source object Y.

By the temporal implications (constraints) inherited from the overlap relation, the intermission period τ should be less than x and $\tau + y$ is larger than x.

X starts Y Figure 24 represents an IPNs model of the synchronization agent which controls the media objects, X and Y, where the playout of the object X starts at the same time as $Y's$.

When a token with the color $< x, Xsource >$ arrives at the place A, simultaneously, a token with the color $< y, Ysource >$ arrives at the place B. This will start producing the control tokens for both X and Y media sources.

By the temporal implications (constraints) inherited from the start relation, $y > x$, makes the playout of the object Y last longer than

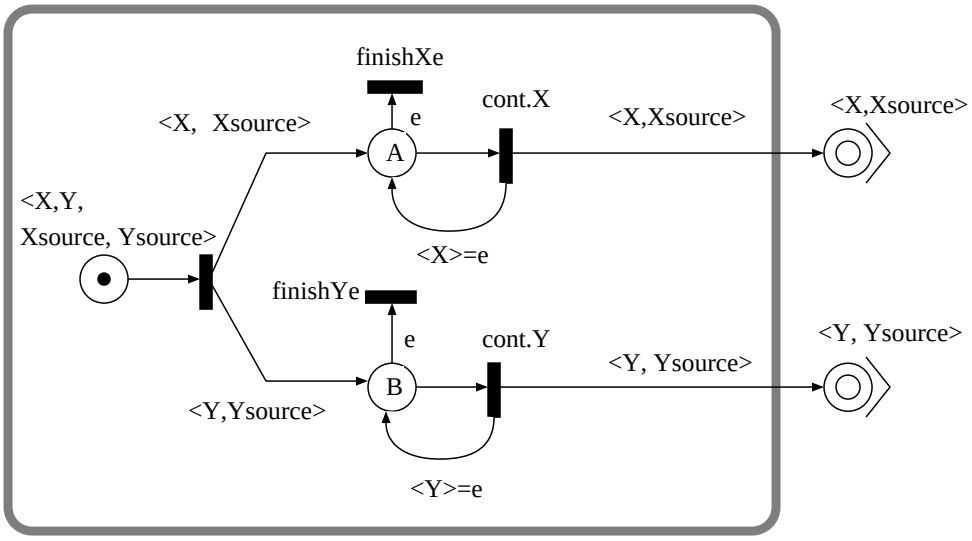


Figure 24: An IPNs model of x (s) y.

that of the object X.

X finishes Y Figure 25 represents an IPNs model of the synchronization agent which controls the media objects, X and Y, where the temporal relation between the playout of the objects X finishes the playout of the object Y (x (f) y).

When a token with the color $< x, Xsource >$ arrives at the place A after finishing the playout of the intermission period, τ, the place B is already producing the control tokens. When the playouts of objects X and Y finish, the transition $finishXeYe$ fires. This enforces the finishing of both playouts at the same time, especially by applying the strict firing rule stated earlier.

According to the temporal implication of the finish relation, $\tau > 0$, this makes the playout of the object Y last longer than that of the X object, but both finish at the same time. It is clear that the IPNs model of 'x (f) y' is similar to that for 'x (d) y'. The only difference is the ending points of objects X and Y.

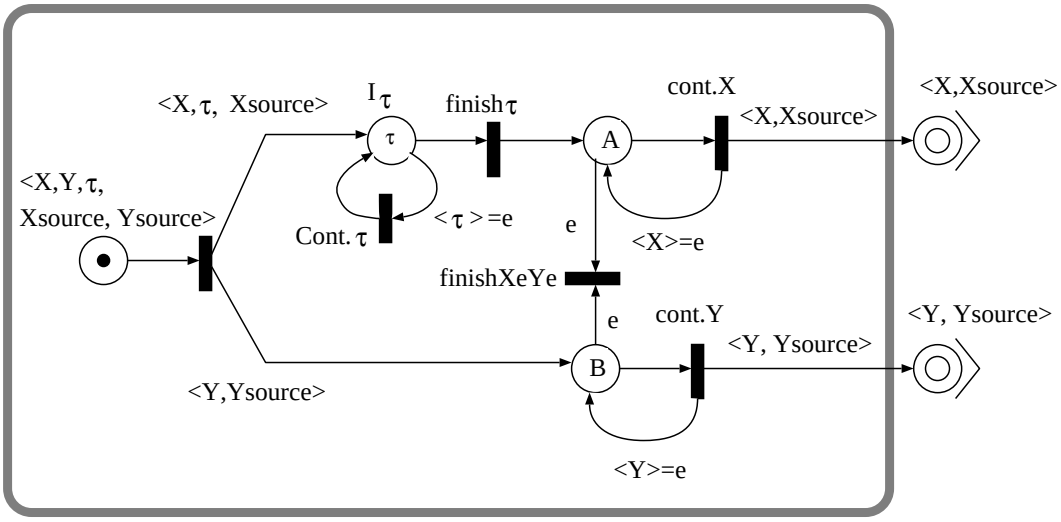


Figure 25: IPN Model of x (f) y.

References

[All83] J. F Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26:832–843, 1983.

[AS94] Yahya Y. Al-Salqan. MediaWare: A Distributed Multimedia Computing Environment and Formalism Based on Interoperable Petri Net. *Ph.D. Thesis, University of Illinois at Chicago*, 1994.

[ASC95] Yahya Y. Al-Salqan and Carl K. Chang. MediaWare: A Distributed Multimedia Environment with Interoperability. *Proceedings of the 4th IEEE Workshop on Enabling Technology: Infrastructure of Collaborative Enterprise-WET-ICE*, April 1995.

[ASCR95] Yahya Y. Al-Salqan, Carl K. Chang, and Y. V Reddy. MediaWare: On Distributed Multimedia Synchronization. *Proceedings of the 2nd IEEE International Conference on Multimedia Computing and Systems*, pages 150–158, May 1995.

[BGV91] W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinement of Petri nets. *Lecture Notes in Computer Science: Advances in Petri nets 90*, pages 1–46, 1991.

[GBC88] A Ghafoor, P. Berra, and R. Chen. A distributed multimedia database system. *Proceedings of IEEE Workshop on future trends in distributed computing systems in 1990s*, pages 461–469, September 1988.

[GC91] J. Gemmel and S. Christodoulakis. Principles of delay sensitive multimedia data retrieval. *Proceedings of first IEEE international conference on multimedia information systems*, pages 147–158, January 1991.

[GL81] H.J. Genrich and K. Lautenbach. System Modeling with High-Level Petri Nets. *Theoretical Computer Science*, 13:109–136, 1981.

[Gro92] Object Management Group. *The Common Request Broker: Architecture and Specification.* Object Management Group and X/Open: Document number 91.12.1, Framingham, MA, 1992.

[Jen81] K. Jensen. Colored Petri Nets and invariant-method. *Theoretical Computer Science*, 14:317–336, 1981.

[Jen91] K. Jensen. Coloured Petri nets: A high level language for system design and analysis. *Lecture Notes in Computer Science: Advances in Petri nets 90*, pages 342–416, 1991.

[LBea90] W.H. Leung, T. J. Baumgartner, and Y. H. Hwang et al. A software architecture for workstations supporting multimedia conferencing in packet switching networks. *IEEE Journal on selected areas in communications*, 8(3):380–390, April 1990.

[LG90] Thomas D. C. Little and Arif Ghafoor. Synchronization and storage model for multimedia objects. *IEEE Journal on selected areas in communications*, 8(3):413–427, April 1990.

[LG91] Thomas D. C. Little and Arif Ghafoor. Multimedia synchronization protocols for broadband integrated services. *IEEE Journal on selected areas in communications*, 9(9):1368–1381, December 1991.

[LKG94] L. Li, A. Karmouch, and N.D. Georganas. Multimedia Teleorchestra with Independent sources: Part 1- Temporal Modeling of Collaborative Multimedia Scenarios. *Multimeda Systems*, pages 143–153, February 1994.

[Mur89] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[Neu93] Phillip D. Neumiller. Optimal Object Oriented Design of Data Networking Software using an Actor Augmented Booch Methodology. *Master Thesis, University of Illinois at Chicago*, 1993.

[Nic90] Cosmos Nicolaou. An architecture of real-time multimedia communication systems. *IEEE Journal on selected areas in communications*, 8(3):391–400, April 1990.

[PR93] B. Prabhakaran and S.V Raghavan. Synchronization models for multimedia presentation with user participation. *Proceedings of first ACM conference on multimedia*, pages 157–166, August 1993.

[QWG93] Naveed U. Qazi, Miae Woo, and Arif Ghafoor. A synchronization and communication model for distributed multimedia objects. *Proceedings of first ACM conference on multimedia*, pages 147–156, August 1993.

[Rei83] W. Reisig. Petri nets with individual tokens. *Informatik-Fachberichte*, 66(21):229–249, 1983.

[Ste90] Ralf Steinmetz. Synchronization properties in multimedia systems. *IEEE Journal on selected areas in communications*, 8(3):401–412, April 1990.

[Ste92] R. Steinmetz. Multimedia Synchronization Techniques: Experience Based on Different System Structures. *Proceedings of IEEE Multimeda'92*, pages 306–314, 1992.

[WJ94] Mi Wooldridge and N. R. Jennings. Agent Theories, Architectures, and Languages: A Survey. *Lecture Notes in Computer Science: Intelligent Agents*, pages 1–30, 1994.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Engineering Review*, 10(2), October 1995.

[WWR95] T. Wahl, S. Wirag, and K. Rothermel. Tiempo: Temporal modeling and authoring of interactive multimedia. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 274–277, Washington, DC, May 1995. IEEE Computer Society.