MODELING ADAPTIVE MEDIA PROCESSING WORKFLOWS

K. Selçuk Candan[#] *Gisik Kwon*[#] *Lina Peng*[#] *Maria Luisa Sapino*[$]

[#]Computer Science and Engineering Dept.
Arizona State University
Tempe, AZ, 85287, USA.
{candan, lina.peng, gisik.kwon}@asu.edu

[$]Dipartimento di Informatica
Universita' di Torino
Torino, Italy
mlsapino@di.unito.it

ABSTRACT

ARIA, ARchitecture for Interactive Arts, is a middleware to process, filter, and fuse sensory inputs and actuate responses in real-time. An ARIA media processing workflow describes *how* the data sensed through media will be processed and *what* audio-visual responses will be actuated. Each object streamed between ARIA processing components is subject to transformations, as described by a media workflow graph. The media capture and processing components, such as media filters and fusion operators, are programmable and adaptable; i.e, the delay, size, frequency, and quality/precision characteristics of individual operators can be controlled via a number of parameters. In this paper, we present the underlying model which captures the dynamic nature of the ARIA media processing workflows. This model enables design time verification, optimization, and runtime adaptation.

1. INTRODUCTION

Interactive performances (which incorporate realtime, sensed, and archived media and audience responses into live performances) require an information architecture that processes, filters, and fuses sensory inputs and actuates audio-visual responses in real-time, while providing appropriate QoS. On the other hand, currently, media artists and choreographers are using best-effort tools, such as Max/MSP [1] and Pd[2], which are limited in their expressive power and lack the capability of describing and enforcing realtime constraints. ARchitecture for Interactive Arts (ARIA) is a middleware we designed to capture, stream, and process various audio, video, and motion data in such sensory/reactive environments.

ARIA media processing workflows are modelled as directed graphs where vertices (or nodes) represent sensors, filters, fusion operators, and actuators, while edges represent connections that stream objects between components (Figure 1). In this paper, we present the underlying model which captures the dynamic nature of ARIA. This model enables us to develop optimization [5] and adaptation [10] algorithms for QoS supported realtime operation. It also enables design time reasoning on the resource-aware and quality-adaptive nature of the workflows and estimation of their runtime properties.

THIS WORK IS SUPPORTED BY NSF GRANT # 0308268

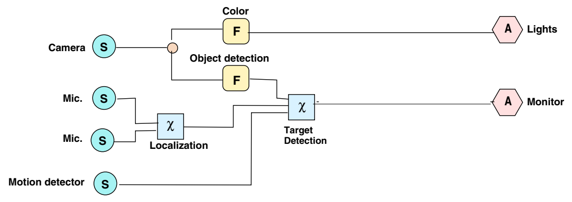

Fig. 1. A media processing workflow: "S" denotes sensors, "F" filters, "χ" fusion operators, and "A" actuators

2. ARIA MEDIA PROCESSING WORKFLOWS

An ARIA media processing workflow is created by combining various ARIA operators (i.e., nodes in the workflow graph) which sense, filter, transform, and fuse media objects.

Objects: The basic information unit is a data object. Depending on the task, an object can be as simple as a numeric value (such as an integer denoting the pressure applied on a surface sensor) or as complex as an image component segmented out from frames in a video sequence. Each object has (a) an *object payload*, such as a string, a numeric value, or an image segment, and (b) an *object header*, which consists of

- an *object property descriptor*, describing the object state. It includes object size and object precision[1].
- an *object history descriptor*, consisting of the set of *resource usage stamps* and *timestamps* acquired by the object's predecessors as they go through various operators. Among other things, the history descriptor enables the computation of the overall *age* of an object in the system (total delay observed since sensing) and the corresponding resource consumption.

Operators: The adaptive nature of ARIA is largely thanks to the components of the architecture that are programmable; in particular, the delay and quality characteristics of individual operators can be controlled via input parameters.

[1]For different object types, precision may mean different things; for an image, its precision may mean its resolution, whereas for a coordinate value, it may mean the level of confidence provided by the object tracking sensors.

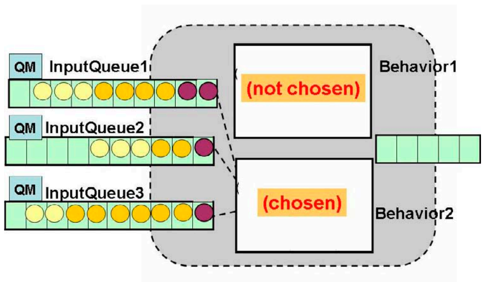

Fig. 2. An operator with three input queues and two behaviors; only one behavior will operate on the selected (darker shaded) input objects. Each queue is overseen by a queue manager, which may shed some of the objects (light shaded)

Sensors act as object sources. A scalable sensor can make objects available at different frequencies, sizes, and precisions. While sensors generate object streams, actuators consume object streams and map them to appropriate outputs.

Filter and fusion operators have rich analysis, aggregation, and filtering semantics. In particular, they may perform complex media processing, information clustering, and data cleaning tasks. A *filter* takes a single stream of objects as input, processes and transforms its inputs, and outputs a new object. For example, consider a module that takes facial images as its input and returns face signature vectors as its output. This module is a transforming filter. Note that the precision of the result may depend on the number of consecutive face images considered or may depend on the algorithm used. Consequently, filters are adaptable and may provide multiple precisions, each with its own delay and resource requirement. A *fusion* operator is similar to a filter, except that it takes multiple input object streams. For example, consider a component which receives object-tracking information from multiple redundant sensors and outputs *fused* highly-precise object-tracking information. Fusion operators are also adaptable. In general, we model an operator as follows (Figure 2):

Definition 2.1 (Operator) *An operator, v, has input and output queues ($Inqu(v) = \{in\_qu_i, 1 \leq i \leq l\}$ and $Outqu(v) = \{out\_qu_j, 1 \leq j \leq k\}$) and a set of behaviors, $B(v)$. The operator picks a set, $pickset_i$ (where $|pickset_i| = w_i$), of objects from each input queue in_qu_i and applies one of its behaviors, $b \in B(v)$, to the resulting l-tuple*

$$input\_intance = \langle pickset_1, pickset_2, \ldots, pickset_l \rangle$$

and outputs processed results into the output queues.
Based on the characteristics of the selected behavior function, $b \in B(v)$, the operator fuses and transforms the object payloads (and combines and updates object histories).

Resources: Let Res denote the set of hardware resources on which the ARIA architecture operates. Each resource, $r \in Res$ has a limit, $limit(r)$, beyond which it stops functioning properly. Example resources include network resources, CPU, and buffer available to ARIA nodes.

Queues: An operator v with l incoming and k outgoing edges has l input queues, $Inqu(v)$, and k output queues, $Outqu(v)$. Each edge $e \in E$ between nodes v_i and v_j connects one output queue of v_i to an input queue of v_j.

Each queue, q, in the system has limit on the number of objects, $obj\_limit(q)$, and a byte limit, $byte\_limit(q)$. Once any one of these limits is passed, no new objects are admitted into the queue until the queue length drops below the upperbound. A queue manager, $QM(q)$, oversees the queue, q.

Overloaded queues may necessitate object shedding [10]. Thus, each queue manager, $QM(q)$, has a drop policy which governs the object shedding criteria:

- drop selection criteria, $dropC$, help choose which objects to drop based on the age, size, precision, or history (amount of resources used so far etc.).

Alternative Behaviors and Implementations: Each operator node v has a set of behaviors, $B(v)$, where a behavior, b, of an operator with l inputs is a tuple $\langle inv\_sig, inqueue\_par_1, \ldots, inqueue\_par_l \rangle$, where

- inv_sig is the invocation signature, consisting of the name of the code (or library) and the set of applicable input parameter values; for instance, an image filter has a different behavior for different distortion parameters. Note that the complexity of the operator (in terms of delay and CPU usage) and the result precision varies with the value of the distortion parameter.
- $inqueue\_par_i$ is a tuple, $\langle w_i, ignC_i, prC_i, \phi_i, sftC_i \rangle$, describing the operation characteristics of the i^{th} queue for this particular behavior. These are described next.

Note that each behavior of a given operator can potentially use the objects in the queues differently. The parameters governing how a behavior uses the queue in_qu_i are as follows:

- a window size, w_i, denotes the number of objects from this queue to be used by the behavior at a time,
- object priority criteria, prC_i, determine the order of the objects in the queue for this behavior. If the criterion is FIFO, objects are read from the queue in first in first out manner. However, objects can also be read in the order of a priority, based on the age, size, precision, or history (such as the amount of resources already consumed).
- object ignore criteria, $ignC_i$, determine which objects in the queue should be ignored by this behavior. This can be based on the age, size, precision, or history.

Once a behavior triggers and starts processing the selected input objects, it may mark some objects in the queue (possibly the objects used as inputs) unavailable for the next cycle.

- a shift parameter, ϕ_i, denotes the number of objects that will be marked unavailable as a result of a triggering.
- object shift criteria, $sftC_i$, determine which objects will be marked unavailable. This condition can be based on the age, size, precision, or history (amount of resources used so far etc.) of the objects in the queue.

A special marker `used` identifies those objects which have been used as input objects if they will be marked unavailable for the next cycle.

The lightest objects in Figure 2 are those objects that are being ignored by the selected behavior. The darkest objects, on the other hand, are those that are selected for processing.

Note that since each operator has multiple behaviors, each object in a queue is accessible by the multiple behaviors of the corresponding node. An object in a queue processed and marked unavailable by one of the implementations may be ignored or not, depending on the ignore criteria ($ignC$) of the other implementations. If an object in the queue is marked unavailable or ignored by all implementations of an operator, the object is dropped from the corresponding queue.

In ARIA, depending on the properties of an incoming object (which in turn may depend on the characteristics of the operators applied in earlier nodes) a node may necessitate a different behavior. As illustrated below, the operating and output characteristics of the queues and the implementation semantics of the operator determines the output properties.

Example 2.1 *Let us consider a motion detector filter which receives a stream of frames and identifies the motion direction of a frame-object in this stream. Let the following describe the operation characteristics of this filter: (a) the filter has a parameter w which denotes how many input frames are used together to identify the motion direction, (b) the filter has a phase parameter ϕ which denotes how many input frames are skipped between two computations, (c) the queue drops one out of every λ input frames. Let us also assume that the frequency of the frames is f_{in} and the size of each frame is s_{in}. We can compute the output characteristics as follows:*

- *since the queue drops one out of every λ input images, the effective input frequency of the filter is $f_{in} \times \frac{\lambda}{\lambda+1}$*
- *the buffer requirement of the filter is $r = w \times s_{in}$,*
- *the filter computes one output for each ϕ input images, therefore its output frequency is $f = \frac{f_{in} \times \frac{\lambda}{\lambda+1}}{\phi}$*
- *the operation delay (the maximum delay observed by an input image before the output is generated) is $d = (w-1) \times \frac{1}{f_{in} \times \frac{\lambda}{\lambda+1}} + d_{processing}(s_{in})$*
- *the output precision is a function of the processing precision of the filter operator as well as the window size, the precision of the input objects, and the rate at which some of the inputs have been dropped from the queue before it is being processed*
- *the size of the output objects, $sizeof(motion\_object)$, is fixed and independent of the input size.* ◇

Output qualities of the behaviors are modeled as *functions* of the assessments of input objects. Each behavior, b, has an associated quality merge function, μ_b, describing the qualities of the output objects in terms of the qualities of the inputs.

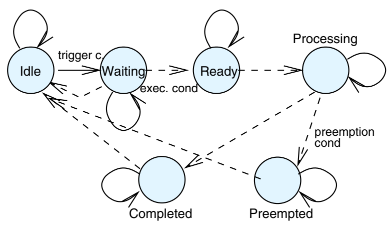

Fig. 3. State transitions for the behaviors of hard operators

In general, each behavior has a fixed amount of processing bandwidth. If the operator is receiving more inputs to consider, then the behavior should choose one appropriate input combination (referred to as $input\_instance$) to process. Such combination selection decisions have significant impacts on the qualities of the fusion results. Therefore, each behavior has an *input selection model,* which answers the question "which combination of inputs in the input queues will be selected as $input\_instance$ to be processed by the behavior?" The way the operator picks its inputs is governed by resource, time, and quality constraints. When the number of combinations to consider is larger than the behaviors capacity, then system needs to shed (not the individual queued objects but) combinations of objects that are not promising fusion candidates. Therefore, each behavior also has an *input combination shedding model.* Finally, the *consumption model* of a behavior determines whether an object, o_i, already included in an $input\_instance$ processed by the behavior, will be reconsidered for further operations or will be marked `used` after its use.

Operating Conditions: In ARIA there are four types of operators: *hard regular*, *soft regular*, *hard irregular*, and *soft irregular*. Regular operators operate with a fixed frequency: *hard regular* operators preempt any preexisting operation if not completed on time, whereas *soft regular* operators wait until the uncompleted operation finishes. Irregular operators do not have associated frequencies; their behaviors are triggered only when certain conditions are satisfied. In general, behaviors are triggered only when one or more of the following trigger conditions are satisfied:

- A *temporal regulating condition (TC)* is satisfied when a proper amount of time has elapsed since the last time the frequency condition was satisfied.
- A *queue condition (QC)* is satisfied when the queue satisfies certain constraints (for instance, when the number of objects in the queue exceeds a threshold)
- *Object property and history conditions (OPC, OHC)* are satisfied when the property and history descriptors of the objects in the queues satisfy certain constraints (for instance, when the precision of an object in the queue exceeds a threshold)
- A *flow control condition (FC)* ensures the formation of desirable workflows. These conditions enable the

dynamic behavior of the ARIA operators by adapting workflows to real-time changes in the environment.

Furthermore, a behavior can be executed only when its execution conditions are satisfied. A behavior may not be in an executable state for various reasons, including (but not limited to) resource shortages. As the queue changes with the addition and removal of objects, the behaviors of a soft operator, thus, can be in one of the five states.

- `idle`: behavior's trigger condition is not yet satisfied.
- `waiting`: behavior's trigger condition is satisfied, but there is no implementation in the executable state.
- `ready`: behavior's trigger condition is satisfied and there is an implementation in the executable state.
- `processing`: the behavior is allocated the resources needed for its operation, collected all the required objects, and is processing.
- `completed`: the behavior places its output objects into the corresponding output queues and returns its resources back to the system.

The behaviors of a hard operator, on the other hand, can also be in a sixth state (Figure 3):

- `preempted`: the behavior has been preempted; thus it returns the resources it holds back to the system.

Solid state transitions in the graph in Figure 3 occur as objects are inserted or removed from the input queues of the behaviors. A single operator functions as a collection of such state transition graphs, one for each one of its behaviors. The exclusive execution of a single behavior among all possible behaviors is ensured through flow conditions (FCs). We are currently investigating a higher-granularity model where a common subset of trigger functions (such as temporal synchronization primitives) are explicitly captured as in [3].

Media Processing Workflows: An ARIA media processing workflow is a directed connected graph, $G(V, E, \beta, C)$:

- V is a set of nodes and E is a set of edges between the nodes on V.
- Let $V_{l\times k} \subseteq V$ be the subset of nodes in V which have l inputs and k outputs. Let also $B_{l\times k}$ denote all behaviors with l inputs and k outputs. Then, $\beta_{l\times k} : V_{l\times k} \rightarrow 2^{B_{l\times k}}$ is a mapping from the nodes in the network to the subsets of behaviors and β is the collection of all mappings applicable to the nodes in V.
- C is a set of temporal and quality contraints that the media processing workflow should enforce.

For example, C may include an end-to-end quality constraint stating that we need to be at least κ confident that the quality of the objects arriving to an actuator is above a lowerbound, $q_{\perp}$. A mapping, $\texttt{res\_map} : V \rightarrow 2^{Res}$, describe to which resources the operators and their input and output queues are mapped. Static optimization [5] and runtime adaptation [10] schemes use this model for QoS supported operation, by selecting appropriate behaviors and input objects to operate on.

3. RELATED WORK

Adaptation has always been a crucial aspect of data flow systems [13, 6]. Recently there has been a number of efforts in composition of multimedia services (SpiderNet [8], SAHARA [12], SPY-Net [15], CANS [7], and Infopipes [4]). In most of these, the goal is to communicate a media object from a source server to a consumer, while the overlay routing nodes provide (mostly application level) services, such as transcoding and mixing. Unlike these works, which focus on the delivery of an object, our focus is to address challenges associated with media processing workflow design, adaptation, and evolution that arise in sensory/reactive environments. In [14] a multi-resource reservation framework for distributed collaborating services is proposed. Q-RAM provides a set of QoS optimization schemes, with discrete QoS options, in the context of video-conferencing [9]. Unlike ARIA, which considers workflows, Q-RAM considers each task as an independent application and imposes the quantification of the system utility on users via weighting the linear combination of all quality dimensions of all applications.

4. CONCLUSIONS AND FUTURE WORK

In this paper, we described the model for representing the dynamic and adaptive nature of operators in ARIA media processing workflow middleware. The model presented here is *memoryful* in the sense that state transition conditions are described through high-level trigger conditions that rely on object histories. We are currently investigating a model where a common subset of trigger functions (such as temporal synchronization primitives) are explicitly captured.

References

[1] http://www.cycling74.com/index.html

[2] http://www-crca.ucsd.edu/~msp/Pd_documentation/

[3] P. Bertolotti, O. Gaggi, and M.L. Sapino, *A State-Transition Model for Distributed Multimedia Documents*, DMS 2004.

[4] A. Black, *et al.*. *Infopipes: An abstraction for multimedia streaming,* Multimedia Systems 8: 406-419, 2002.

[5] Lina Peng, *et al. Optimization of Media Processing Workflows with Adaptive Operator Behaviors* MTAP Journal, 2006.

[6] D.Carney*et.al. Monitoring Streams-A New Class of Data Management Applications.*VLDB02.

[7] X. Fu, W. Shi, A. Akkerman and V. Karamcheti. *CANS: Composable, Adaptive Network Services Infrastructure,* USITS 2001.

[8] X. Gu and K. Nahrstedt. *Distributed Multimedia Service Composition with Statistical QoS Assurances,* IEEE Trans. on Multimedia, 2005.

[9] C. Lee *et al. On Quality of Service Optimization with Discrete QoS Options,* IEEE Real Time Tech. and App. Symposium, 1999: 276-.

[10] Lina Peng and K. Selcuk Candan. Confidence-driven Early Object Elimination in Quality-Aware Sensor Workflows, DMSN 2005.

[11] L. Peng *et al. ARIA: An Adaptive and Programmable Media-flow Architecture for Interactive Arts,* ACM MM Inter. Arts Program, 2004.

[12] B. Raman and R. H. Katz. *An architecture for highly available wide-area service composition,* Computer Communication, 26(15):1727–1740, September 2003.

[13] M.A.Shah and S.Chandrasekaran, *Fault-Tolerant, Load-Balancing Queries in Telegraph,* SIGMOD Record, v.30 n.2, p.611, June 2001.

[14] D. Xu, K. Nahrstedt, and D. Wichadakul, *QoS and Contention-Aware Multi-Resource Reservation,* Cluster Computing, 4, 95-107, 2001.

[15] D. Xu and K. Nahrstedt. *Finding Services Paths in a Media Service Proxy Network,* MMCN, 2002.