ARIA: An Adaptive and Programmable Media-flow Architecture for Interactive Arts *

Lina Peng K. Selçuk Candan Kyung D. Ryu Karamvir S. Chatha Hari Sundaram
Computer Science and Engineering Dept.,
Arizona State University, Tempe, AZ, 85287, USA.
{lina.peng, candan, kdryu, kchatha, hari.sundaram}@asu.edu

ABSTRACT

We are developing an adaptive and programmable media-flow ARchitecture for Interactive Arts (ARIA) to enable real-time control of audio, video, and lighting on an intelligent stage. The intelligent stage is being equipped with a matrix of floor sensors for object localization, microphone arrays for sound localization, beam forming and motion capture system. ARIA system provides an interface for specifying intended mappings of the sensory inputs to audio-visual responses. Based on the specifications, the sensory inputs are streamed, filtered and fused, actuate a controllable projection system, sound surround and lighting system. The actuated responses take place in real-time and satisfy QoS requirements in live performance. In this paper, we present the ARIA quality-adaptive architecture. We model the basic information unit as a data object with a meta-data header and object payload streamed between nodes in the system and use a directed acyclic network to model media stream processing. We define performance metrics for the output precision, resource consumption, and end-to-end delay. The filters and fusion operators are being implemented by quality aware signal processing algorithms. The proper node behavior is chosen at runtime to achieve the QoS requirements and adapt to input object properties. For this purpose, ARIA utilizes a two-phase approach: static pre-optimization and dynamic run-time adaptation.

Categories and Subject Descriptors: J.5 [Arts & Humanities]: Performing arts; C.3 [Special-purpose & Application-based Systems]: Real-time and embedded systems

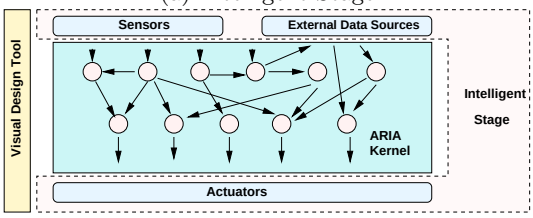
General Terms: Design

Keywords: Tools for creating multimedia art, interactive, multi-model art

1. INTRODUCTION

An Intelligent Stage is being built to detect and classify the movement of the performers and respond by changing environmental elements to achieve spontaneous lighting and sound effects, allowing performers to have real-time control of the stage. The stage in Figure 1(a) is being equipped with a matrix of floor sensors for object localization and motion classification, microphone arrays for sound localization and beam forming, Vicon 8i Realtime system with 8 cameras for tracking the position and motion of human body that is marked with reflective markers, ultrasound sensors for object tracking, and video sensors for determining the presence of marked persons and their relative spatial positions. The researchers from multiple disciplines work together to extend the current functionality of the Intelligent Stage by including novel sensing, interaction, and response mechanisms. The extension requires an innovative information architecture that processes, filters, and fuses sensory inputs and actuates audio-visual responses in realtime. Therefore, we develop an adaptive and programmable ARchitecture for Interactive Arts (ARIA) which will enable design, simulation, and execution of interactive performances.

(a) Intelligent Stage



(b) ARIA

Figure 1: (a) The Intelligent Stage; (b) ARIA: A high level overview of ARIA architecture

ARIA is designed to capture and stream various types of audio, video, and motion data, access external archived data sources, extract various features from streamed data, fuse and map streams onto output devices, and satisfy the QoS requirements. Figure 1(b) illustrates a high-level overview of the ARIA architecture in the theatrical environment. Three

*This work has been supported by NSF Grant # IIS-0308268

MM'04, October 10-16, 2004, New York, New York, USA.

theatric scenarios of ARIA application are generated through collaboration among Intelligent Stage artists, choreographers, and media researchers:

- *Interactive Movement Poses.* Two performers, an adult and a child performers, are tracked by a 3D-motion tracking device and their locations on the stage are tracked using pressure sensors. ARIA continuously monitors the position of the body markers of performers in 3D space, the positions of performers and details of their gestures. The output of the 3D motion tracking is filtered through ARIA to obtain the shape and degree of confidences of the pose. For instance, the adult performer is scripted to draw shapes in the air with his arm. The shapes are recognized by ARIA with the degree-of-confidence and the recognized shape will drive the image from media repository to be projected on the stage. The degree of confidence of recognition and relative spatial locality of the two performers will be fused to generate the color pattern of images to be projected. The image and color information is transferred to the projection actuators.
- *Interactive Sound and Motion.* Besides the 3D-motion tracking device and pressure sensors, the adult and child performer are tracked by a microphone-array as well. Location information from the pressure sensors is used to tune the microphone array for high-quality sound capture. The captured voices of the performers are filtered to identify elevations in amplitude and pitch. ARIA filters pressure sensor information to identify gesture boundaries of the motion of the performers with the stage space. The elevation information is fused with the gesture boundary information for processing synchronized play-back and pre-recorded sound in real-time.
- *Interactive Movement Poses with Sound and Motion.* Furthermore, ARIA fuses and synchronizes the image projection from Scenario 1 with sound playback from Scenario 2 based on the interaction mechanism, like dove-tailing, concurrent or interrupt.

1.1 Related Work

Max/MSP [1] is an object-oriented graphical programming environment for music and multimedia in the MIDI standard format. Programs or patches are created by connecting objects together with patch cords. Events, which could be anything from a MIDI note to a mouse click, flow through the patch cords from one object to another. Jitter extends Max/MSP with a set of media objects for the Max graphical programming environment. Pure Data (Pd) is a real-time graphical programming environment for audio and graphical processing [2]. MAX and Pd provide a visual language framework to describe the connectivity amongst sensors, media filters, media fusion operators and actuators. GrIPD [3] provides a cross-platform extension PD, which allows the design of custom user interfaces for PD patches. CPS [4] is an interactive programming environment for audio. Like MAX and Pd, it provides a visual interface to build interactive sound installations. Image/ine [5], on the other hand, provides an environment for bringing digital video into performances. However, these software are limited in their expressive power and lack the capability of describing realtime application constraints and alternative operator properties. Consequently, they cannot take advantage of alternative media-flow strategies to provide realtime and Quality of Service (QoS) guarantees. In effect, they are best-effort solutions, where the use is limited to very specific hard-coded scenarios.

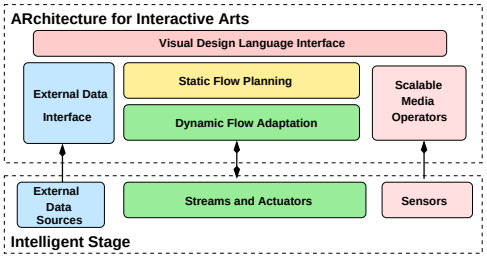


Figure 2: ARIA Architecture

In the past few years, there has been a flurry of activities in the area of data stream processing. Telegraph [6] provides an adaptive dataflow engine that moves data through a number of database operators; STREAM [8] supports continuous queries on the data which has the continuous nature; Aurora [7] focuses on QoS and memory- aware operator scheduling and semantic load shedding for coping with transient spikes in a data stream network. The existing work is focused on managing continuously arriving tuples where tuple sizes are linearly modified by traditional database operators. ARIA models the basic information unit as an object. The design and implementation of ARIA operators are scalable and embed the QoS support. The ARIA filters and fusion operators are implemented by quality aware signal processing algorithms. These operator characteristics are constrained by a collection of precision, resource, and delay specifications. The output precision, consumed resource and end-to-end delay of an object depend on the property of operators on the object delivery path. Different implementations of an operator are leveraged by ARIA Optimizer to deliver high precision, small delay, and small resource-usage objects to the actuators.

In the following sections of the paper we describe ARIA architecture, the network processing model, the performance metrics, and ARIA optimizer in Section 2. In Section 3, we show how well the alternative path identification experimentally approximates the optimal.

2. ARIA ARCHITECTURE

Figure 2 illustrates the components of ARIA. We first provide an overview of main components of ARIA architecture.

The basic information unit in transmission is a *data object*. An object is produced by a sensor or an external data source. 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 streamed through ARIA contains an object payload, such as a string, a numeric value, or an image region, and a meta-data header, describing the object properties. Each *stream* denotes a transmission channel of objects of the same type. Sensors

(a) (b)

Figure 3: (a) An example flow network: "S" denotes sensors, "F" denotes filters, "χ" denotes fusion operators, and "A" denotes actuators; (b) the corresponding flow network created in *qStream* GUI (note that GUI shows that some of the nodes have multiple/alternative implementations)

act as stream sources. While *sensors* generate object streams, *actuators* consume object streams and map them to appropriate outputs. *Scalable Media Operators*, such as filter and fusion operators, are implemented by quality aware signal processing algorithms. A *filter* takes an object stream as input, processes and transforms the objects in the stream, and outputs a new stream consisting of the transformed objects. For example, consider a module that takes a stream of facial images as its input and returns a face signature vector as its output. A *fusion operator* is similar to a filter, except that it takes as its input multiple streams and returns as its output multiple streams. For example, consider a module which receives object-tracking information from multiple redundant sensors and outputs *fused* highly-precise object-tracking information. For a specific signal processing task, it is often the case that there exists tradeoffs between desired characteristics, such as the delay and perceptual distortion/precision. For example, consider MPEG decoding of interlaced video sequences. The maximum precision implementation of the well-known IDCT operation requires all coefficients. A slightly lower precision and lower delay implementation can be achieved by setting a few high-frequency coefficients to zero.

The main task of ARIA *Optimizer* is to explore the potential design space to generate the near-optimal implementation of algorithms. A *media-flow plan* is the term that we use to specify which implementation of a single algorithm shall be executed by ARIA runtime kernel. ARIA Optimizer takes place in two stages. At the first *static flow planning* stage, ARIA Optimizer generates a set of static media-flow plans through graph-theoretical heuristic and approximation algorithms to obtain the least-delay high-precision output streams (Section 2.2). Due to static nature of the processing delay and resource usage models, plans produced by the static optimizer may not be optimal at runtime. At the second *dynamic flow adaption* stage, ARIA Optimizer dynamically makes decisions on the selection of media-flow plan alternatives that are registered at the first stage. At this stage, each operator uses information about the delays and precisions of other operators in the network to adapt to real-time situations. Since it is not efficient to make global system state available to each operator in the system, a relevant subset of global information is identified and made available for each operator.

External Data Interface provides access for external sources, including relational and multimedia databases as well as Internet sources for the proper operation of some filter and fusion operators, like image-matching filters. Prefetching/view management techniques to ensure that the relevant information will be available when needed are being investigated.

Visual Design Language Interface provides the choreographer with a visual design tool to describe the processing operators in the interactive performance. The specifications include streaming characteristics of the sensors; precisions and computational overheads of the feature extractors; schemas, interfaces, and data access costs of the external data sources; functionality and QoS of the fusion operators and integration pathways in the media-flow network; display and audio features of the actuators. The interface also provides mechanisms for describing local (per-processing-component) and end-to-end (sensor-to-actuator) delay, QoS, and frequency constraints.

2.1 The Network Processing Model

The basic job of ARIA is to filter sensor data and fuse them to produce the desired interactive media effects. It is natural to model ARIA media-flow processing in the form of networks. Hence, we model sensors, filters, fusion operator, and actuators as a set of nodes and media object streams as arcs that connect processing operators. Figure 3(a) illustrates one example of ARIA media-flow processing network. *qStream* is the system that provides a graphical interface for the choreographers to specify a valid flow processing network based on their tasks. Figure 3(b) shows the corresponding flow network created in *qStream* GUI.

Formally, we define ARIA media-flow network as an acyclic, directed connected network, $G(V, E, \beta)$, where

- the set, V, of vertices(or nodes) represents the set of sensors, filters, fusion operators, external sources and actuators;
- the set, E, of edges represents object and feature streams between components;
- the set, β, of behaviors describes operation characteristics of the nodes. A behavior of a node, $b = \{< s, r, f, p, d >\}$, is a collection of size, resource, frequency, precision, and delay behaviors, respectively. As in Figure 3(b), each node may have several alternative scalable implementations each of which is modelled by an alternative behavior.

The precision, resource requirements, and delay characteristics of the ARIA operators depend on the parameter assignments as we discussed in Section 2. In other words, individual

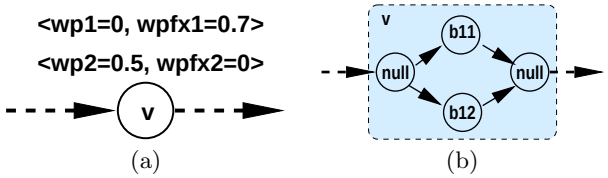


Figure 4: (a) A node with two precision behaviors; (b) the ABNs of the node

operator performance is controllable through the assignment of the parameter values. The assignment of the parameter values can be fulfilled either by the user or by the ARIA optimizer. Given a flow graph $G(V, E)$, the assignment of the parameter values can be denoted as $A(\nu)$, where $A(\nu)$ is the parameter assignment for the operator corresponding to vertex ν. Using the assignment values of the ARIA operators, we develop the performance model of outputs as following:

- Precision: Precision denotes the *quality* of an object. Precision is a multiplicative metric. Given an operator, the output precision is based on the precisions of the input objects and the precision behavior of the operator.
- Delay: Delay is additive in nature. Given an operator, the delay is the time interval that is needed by the operator to process the object. Given an object, the total end-to-end delay observed by this object can be calculated by adding up the delay introduced by the operators that output the object plus the maximum delay observed by the input objects used by the operator.
- Resource Requirement: Resource requirement is also additive in nature. However, in contrast to the delay, the total amount of resources required for an object can be calculated by adding the resources needed at the operator that outputs this object plus the resources needed to compute each input objects used by the operator.

2.2 ARIA Optimizer

The goals of ARIA Optimizer include: (a) The precision of the final object is maximized; (b) The total resource requirements for streaming an object is minimized; (c) The end-to-end delay observed by an streamed object is minimized. There exist tradeoffs between desired characteristics of a single operator, such as the delay and perceptual distortion. The trade-offs at the operator level result in trade-offs between the performance objectives. Since optimization problems with trade-offs between objectives are generally intractable, we do not expect polynomial time solutions for the quality- and delay/cost-based plan selection problem.

As defined in Section 2.1, in a media-flow processing network, each operator represented as a node can have alternative behaviors. In the graph, each vertex is called an alternative behavior node (ABN) as it represents a single alternative behavior of the corresponding operator. Figure 4(b) shows the two ABNs that correspond to the single flow node shown in Figure 4(a). The problem of selecting an alternate plan in original network is then transformed to select an optimal path from a set of edge disjoint paths. Most shortest path algorithms, like Dijkstra's shortest path algorithm, rely on the observation that "each subpath of a shortest path is a shortest subpath" and use this observation to eliminate non-promising paths. This enables these algorithms to identify shortest paths very efficiently. Unfortunately this condition does not always hold for the various behaviors of the media-flow processing network. While dealing with multimedia objects in various sizes, the effect of object size changes on the transmission cost and resource consumption has to be considered in optimization. In this work, a new class of algorithms are developed to find the optimal execution plan. These operator-specific information include shortest delay paths, high precision paths and spanning trees for streams that have to be delivered to multiple actuators.

3. OVERVIEW OF THE RESULTS

We created a data set corresponding to various media-flow networks. We varied the network properties. In order to observe and interpret the performance of the proposed optimization mechanisms, in addition to our heuristics, we used two alternative optimizations: non-linear mixed integer formulation of the problems (NLMIP) optimized using LINGO and bruteforce enumeration of all alternatives. The bruteforce enumeration enabled us to obtain not only the best solution, but also the worst and average solutions, allowing us to observe the relative performance of our techniques. The experiments showed that the proposed heuristic algorithms indeed work efficiently (under 100ms) and effectively (within 1-2% of the optimal). They scale better than other formulations of the same problem, yet, return almost optimal results within a time-frame suitable for real-time adaptive systems.

4. CONCLUSION

In this paper, we have presented an adaptive and programmable real-time media-flow architecture, ARIA, to support interactive sensing/actuating environments. It offers a new medium which will allow artists to integrate novel sensing, interaction, content, and response mechanisms in stage performances. Thus, enabling digital media and art researchers to expand the expressive and interactive possibilities in performative, multimedia environments. ARIA operators are constrained by the QoS and correspond to quality-adaptive computing elements in the media-flow network. ARIA is designed to extract features from streamed data, fuse and map streams onto output devices, and satisfy the QoS requirments, using these quality-adaptive operators.

5. REFERENCES

[1] http://www.cycling74.com/index.html
[2] http://www-crca.ucsd.edu/~msp/Pd_documentation/
[3] http://crca.ucsd.edu/~jsarlo/gripd/
[4] http://www.bonneville.nl/cps/index.html
[5] http://www.image-ine.org/
[6] M.A.Shah and S.Chandrasekaran, *Fault-Tolerant, Load-Balancing Queries in Telegraph,* SIGMOD Record, v.30 n.2, p.611, June 2001.
[7] D.Carney *et.al. Operator Scheduling in a Data Stream Manager.* VLDB 2003.
[8] S.Babu and J.Widom. *Continuous Queries over Data Streams.* SIGMOD Record, Sept 2001.