Extended Format Definition and Quality-driven Format Negotiation in Multimedia Systems

Marco Lohse, Philipp Slusallek, Patrick Wambach

Computer Graphics Lab, Saarland University,
P.O. Box 15 11 50, 66041 Saarbrücken, Germany
{mlohse, slusallek, wambi}@graphics.cs.uni-sb.de

Abstract. Multimedia middleware needs to support a wide variety of devices together with their respective data formats. This becomes increasingly relevant and difficult in a distributed environment where new devices and formats can become available at any time and must be taken into account when deciding how to set up a flowgraph of distributed multimedia components.
In this paper we present an automatic algorithm for configuring and connecting a high-level flowgraph of multimedia components. Given this abstract flowgraph of participating devices and key components, the algorithm automatically selects necessary additional components, chooses suitable formats, and connects the flowgraph, while trying to achieve the best possible quality.

1 Introduction

Traditional multimedia middleware such as DirectShow from Microsoft [1] or Apple's Quicktime [2] provides access only to multimedia devices directly connected to the same PC an application is running on. The network is supported as a source of data but the middleware cannot extend its control to other devices on the network. For example, it is impossible to transparently watch TV via the TV card in your colleague's PC, or control remote cameras attached to other PCs or directly to the network.

With the increased deployment of networked and mobile multimedia devices, support for such distributed multimedia operation is becoming increasingly important. We are currently developing a distributed multimedia middleware that takes advantage of existing architectures for multimedia middleware and transparently extends them to the network. Of course this requires the redesign of some fundamental services such as synchronization, device control and data transport.

Such a distributed multimedia system must allow a user to create, configure, and manage a complex flowgraph without having to deal with low level details such as data formats and how certain devices must be connected to be compatible with others. New devices and associated new data formats may become available to a system at any time and should be seamlessly supported by the multimedia middleware and thus all of its applications.

For our purposes the term "device" includes hardware devices as well as new software components, including new compression algorithms, network protocols, and file formats. We define "format" as the metadata that describes a multimedia data stream. This includes such properties as sampling rate, resolution, color space, protocol encapsulation and many others. Associated with each of these parameters is a "quality" specified with a weighting function, where a high value expresses a preference for a certain parameter value. A more detailed definition is given later in this paper (see Section 4.1).

New formats from components at the edges of a multimedia flowgraph, such as data sources (e.g. cameras) and sinks (e.g. displays), can often be handled by providing suitable conversion algorithms to formats already supported by the middleware. However, format decisions at the edges can have global effect on a flowgraph as they can propagate along a media flow. A common example is a preferred audio sampling rate of an output device, which should be used throughout the flowgraph in order to avoid sampling rate conversion and its associated problems.

New components that will be used as internal nodes in a flowgraph are more challenging as they can have more impact on the optimal data processing. For instance, optimally using a newly added hardware device that allows efficient DV-to-MPEG transcoding might require re-routing of data across the network to the transcoder instead of using local software transcoding.

Existing systems either require the user to choose and specify the details of the complete flowgraph or rely on a simple template mechanism for selecting missing flowgraph components. Such simple techniques work for the traditional environment with a restricted configuration space where new devices are added infrequently. However, they are inadequate for the dynamic environment of distributed multimedia operation.

In this paper we propose a fully automatic approach to selecting and configuring necessary components for an incomplete flowgraph specified by an user or an application. Given the set of all available multimedia components and their input and output formats, the system uses a modified version of Dijkstra's algorithm for solving the Single-Source-Shortest-Path (SSSP) problem to find the possible connections between two components of the user graph. Additionally this connection should provide the best quality. In order to support complex user graphs the local SSSP algorithm operation is augmented with a global propagation component that forwards local decisions along the chosen paths. Finally, possible solutions are evaluated depending on their computational costs and a complete and valid multimedia flowgraph is created.

We start the paper with a brief review of current systems and previous work in the next section. In Section 3 and 4 we define our environment for specifying multimedia flowgraphs from individual components and their associated data formats. The format negotiation algorithm is then presented in Section 5 with an example given in Section 6.

2 Background and Previous Work

Microsoft's DirectShow [1] is probably the most widely used multimedia middleware. It uses the common approach of providing a large set of small components (called Filters) to be connected in the form of a flowgraph that operates on the multimedia data streams. DirectShow classifies multimedia formats by major type, minor type, format type and optional parameters. The major type includes audio, video and stream formats; the minor type contains extended information about the major type, like the specific YUV format. The local operating system contains a registry of all filters, which allows enumeration of all filters with certain format requirements. In the later versions of DirectShow a simple template mechanism is used for augmenting incomplete flow graphs. As mentioned above this mechanism is insufficient for dynamic and distributed environments.

SGI's Digital Media Library [3] provides a more advanced definition of multimedia formats that is organized in a shallow hierarchy where individual parameters are stored in a list. It provides functions for common operations on formats, such as equality. However, SGI's middleware that is the basis for the new cross platform "open multimedia library (OpenML)" [4] does not provide any automatic mechanism for connecting flowgraphs.

Apple's Quicktime [2] provides an automatic data transform mechanism, which converts input data automatically into the preferred format of a media codec but does not provide a general flowgraph mechanism. Sun's Java Media Framework [5] offers a set of methods to compare formats. A breath-first search algorithm is employed to set up flow graphs but no criterion like costs or quality is used to find a global optimum solution. In summary, all multimedia frameworks are restricted to querying format properties of media processing components and the application is responsible for establishing optimal component connections.

Home networking middleware like HAVi [6], Jini [7] or UPnP [8] offers sophisticated device discovery mechanisms, but only HP JetSend technology directly introduces a concept for format negotiation between components [9]: devices store a preferred encoding format out of their list of supported formats. In a negotiation step, the receiving device is responsible for choosing the encoding. Again, this local model is inadequate for more general media processing.

Little research has been done on application-level format negotiation as described above. However, many algorithms have been developed for automated QoS negotiation in distributed multimedia systems, using for example an agent-based approach [10] or an optimization approach [11]. Rothermel et al. [12] proposed a QoS and format negotiation protocol, which supports arbitrary multimedia flowgraphs. Yet, this approach is limited to determine the QoS parameters of an already fully connected flowgraph. Our approach focuses on augmenting incomplete flowgraphs by automatically adding necessary components, while at the same time trying to achieve the best possible quality and configuring the flowgraph accordingly. Computational costs are only considered to decide between two or more solutions with equal quality.

3 Basic System Design

We follow a common approach for specifying multimedia flowgraphs: devices (e.g. a camera) and software components (e.g. a converter) are represented as *nodes*, which potentially have several multimedia data inputs and outputs; in our case called *jacks*. By connecting input and output jacks, complex flowgraphs can be built. Associated with each jack are one or more supported *formats*. A format provides the metadata that describes the multimedia data to be streamed across a jack. The format negotiation procedure must select exactly one format for each jack and configure all of its parameters. More details on formats are given below in Section 4.1.

Our system distinguishes between six different types of nodes: A *source* produces data and has one output jack. A *sink* consumes data, which it receives from its input jack. A *filter* has one input and one output jack. It only modifies the data of the stream and does not change its format or format specific parameters. A *converter* also has one input and one output jack but can change the format of the data (e.g. from raw video to compressed video) or may change format specific parameters (e.g. the video resolution). A *multiplexer* has several input jacks and one output jack; a *demultiplexer* has one input jack and several output jacks.

There are two different types of dependencies which have to be modeled: First, a converter, multiplexer or demultiplexer might only support certain combinations of input and output formats. We use a concept called *IO-partner*: Each input format is associated with one or more output formats and vice versa. Furthermore, during format negotiation, format parameters will be changed to reflect the actual negotiation process, for instance, a video filter which can process any kind of x- and y-resolution has to reflect the actual chosen video resolution of its input format in its output format.

4 Format Definition

4.1 Format Classification and Specification

A sophisticated format classification and specification mechanism is necessary to support automated format negotiation. The goal of the *format classification* is to divide all possible multimedia formats into possible main categories; in our approach, formats are classified by *type* and *subtype*. The *format specification* then states more precisely the actual parameters within each category. Each parameter is described by *key* and *value*. For example, to completely specify a format like *video/raw*, additional information is needed, like framerate, resolution and color space. The values of parameters are stored as a *set of values*, or as a *set of ranges* (non-intersecting). If no constraints are given on a certain parameter, a *wildcard*-attribute can be used. For example, a display node might support RGB color space (set with one entry), framerates of 1 to 30 fps (set with one range), and arbitrary video resolutions (wildcard).

When trying to connect an output jack to an input jack, one has to check for compatible formats. A *intersection format* of two formats exists if their major and minor types are equal, all parameter keys in one format exist in the other format and vice versa, and all intersections of the ranges of values of corresponding parameter keys are a non-empty set. The existence of an intersection format is called a *match*.

4.2 Extending the Format Definition

In order to perform a quality driven format negotiation, the definition of a format has to be extended. Each entry in a set of possible parameter values is assigned a weight reflecting its respective quality, e.g. a framerate of 30 fps receives a higher quality value than a framerate of 15. This weight is in the range of 0 to 1. For values which are specified as a range, the weights of the lower and upper limits have to be specified. Wildcards, which are not resolved by creating intersection formats, are assigned maximum weight. The weight of a format is computed as the average of its parameters' weights. An additional weighting factor for each parameter can be used to stress the importance of a particular parameter. The weight of an intersection format is determined accordingly.

As mentioned above, a quality driven format negotiation is performed which only takes "costs" into account if two or more solutions with the same quality exist. Since the total costs of an operation performed inside a node depends not only on the type of operation but also on the used input and output formats, we need three different measurements: the input and output costs associated with a format and its specific parameter values and the costs of the operation itself. Since all these costs strongly depend on the chosen values of parameters (e.g. the video resolution), the costs have to be updated during negotiation. While cost measurement is an interesting topic, it is not addressed in the context of this paper.

5 Format Negotiation

After introducing the basic elements of the underlying model, we now define the format negotiation problem: *Given a high-level description of a desired multimedia flowgraph (which is independent of physically possible connections and formats), try to find a solution which provides the highest quality and meets defined requirements. If two or more possible solutions are found, find a cost optimal solution.*

We will give an overview of our approach for solving the format negotiation problem. Then, we will describe this approach in detail and point out the interesting aspects, such as computational complexity.

5.1 Overview

The format negotiation is performed in three steps: First, the *user graph* has to be specified. Basically, this graph only stores the structure of the desired multi-

media flowgraph. It contains the wanted sources, multiplexers, demultiplexers, filters, and sinks, and connections between these; converters can but do not have to be added. The format negotiation will create a valid connected multimedia flowgraph. Although the user graph is independent of formats, additional information can be associated with connecting edges. For example, a certain framerate or resolution of a video stream can be specified. If no additional information is given, the format negotiation method described below will try to find the solution with the best quality. There are several ways to obtain the user graph. A user of the system could model a desired configuration; the application could provide a user graph; or, finally, for simple examples with only sources and sinks, the graph could be generated automatically.

The *format negotiation graph* is a representation, which is derived from the user graph and which is used in the second step, the format negotiation. With this extended representation, several sub-negotiations, or requests, are performed to find an optimal solution in respect to the desired parameter, e.g. quality, and format dependencies as described in Section 3. This step uses a modified algorithm for solving the shortest-path problem. Afterwards, solutions from different requests have to be merged. In this step values of format parameters are set globally. Solutions which provide the same quality, are ranked according to their costs. The following section describes the format negotiation in detail.

Finally, the *result flowgraph* can be created. This flowgraph reflects the user graph in the way that the structure of both is the same, but additional necessary converters are included into the result flowgraph and the individual nodes and connections are properly configured in terms of formats.

5.2 Format Negotiation Procedure

As the edges in the user graph can be specified independently of possible connections (in the sense of matching output and input formats), this graph can not be used for format negotiation. An extended representation, the format negotiation graph, is needed. It contains all nodes of the user graph, but all former edges are removed because they do not necessarily represent possible connections. Instead, all connections which represent a match are added as new edges (see Section 4.1). These edges are denoted with the appropriate intersection formats. It is possible that no new edges might be added during this step, so nodes could remain unconnected. This is the case in the example of Figure 1(a): both devices support the same resolutions, e.g. 640x480 and 800x600, but the camera produces video data in YUV format, whereas the display is only able to present RGB video frames. The only way to connect these two devices is to insert a matching converter between them.

Therefore, in the next step, all existing "matching" converters are inserted into the graph[1]. Then, all possible edges are created and denoted with intersection formats. This includes edges from already existing nodes to converters and

[1] This implies having a node registry mechanism, where all existing converters are stored with their possible input formats.

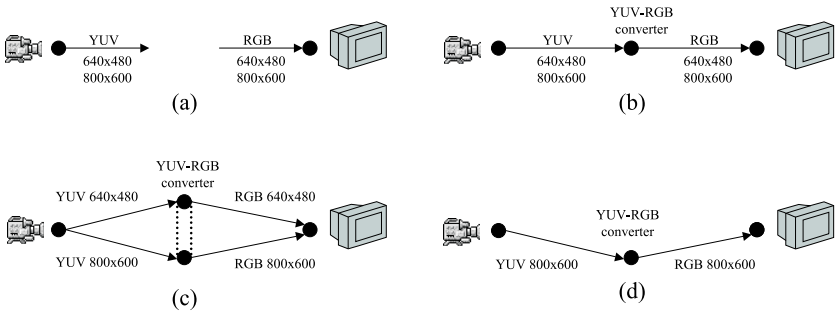

Fig. 1. Basic format negotiation procedure.

vice versa, but also edges from converters to converters. Figure 1(b) shows the result of this step for our example. For simplicity, only one converter is added, as this is capable of performing the required conversion and processing video frames of arbitrary resolution.

Since converters might support several input and output formats, it is necessary to take possible dependencies into account. For example, only specific combinations of input and output formats are allowed (see *IO-partner* in Section 3). Thus, converters are split in the format negotiation graph: for each supported combination of input and output formats, its respective node is split. Since the used format definition allows sets of values as described in Section 4.1, splitting is not only performed per IO-partner, but also per entry in the set of supported parameter values. Figure 1(c) shows the result of the splitting operation.

Due to the fact, that all edges can be labelled with a weight which represents the quality of the corresponding format (as defined in Section 4.2), the format negotiation problem can now be treated as a graph problem. We are interested in finding the path which maximizes the lowest weight on it. This edge is called the "bottleneck" because it determines the overall quality of the solution. Since all weights are positive, one can apply a modified version of Dijkstra's algorithm for solving the Single-Source-Shortest-Path-Problem (SSSP) [13]. Only the initialization function and the relaxation function, which "guides" the search, have to be modified [14]: First, the starting node is labelled with infinite weight, all other nodes with 0. During execution of the algorithm, these values are updated with the current value of the "bottleneck" of the paths leading to this node. The running time of Dijkstra-algorithm is $\mathcal{O}(n\ log\ n+m)$ with n denoting the number of nodes and m the number of edges in the graph. Figure 1(d) shows the result of this step; the higher resolution was chosen because it provides a better quality which is reflected in a higher weight. While a single execution of the algorithm solves the format negotiation problem for simple graphs like the one in Figure 1, in graphs with many multiplexers and demultiplexers a m:n-relationship between sources and sinks is given. That is why we use a divide-and-conquer approach, by dividing the graph into subgraphs. Figure 2 shows the different

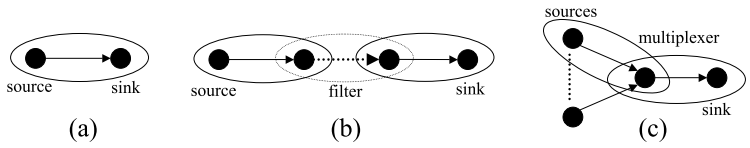

Fig. 2. Different subgraphs.

types of subgraphs. Subgraphs with only one source and sink can be handled with one execution of the algorithm (a). In subgraphs with filters, each filter is first treated as being a sink, and then during the next request as being a source (b). Each input of a multiplexer is treated as a source; the multiplexer itself as a sink (c). Subsequently, the multiplexer is treated as a source. Demultiplexers are treated in a similar way.

Since a solution for one subgraph may lead to an unresolvable conflict or a solution with lower quality later on, all possible solutions for all subgraphs have to be computed. Fortunately, this number is very small in practice, since it is only influenced by the number of converters for a specific format. The results of all requests are then combined to obtain a valid solution. Again, the number of possible combinations is very small because typically only a few solutions for subgraphs can be found which fit together. As parameters are set at this stage, possible dependencies as described in Section 3 have to be taken into account. A general solution which can be used for arbitrary graphs would be the protocol described in [12]. In the special case when there is one edge through which all multimedia data flows, another approach can be used. This includes graphs with one source or one sink only, but also more advanced graph layouts with many sources, sinks, multiplexers and demultiplexer, as long as there exists a "narrowest" location in the graph. Starting from this location, all parameters can be set by simply propagating them upstream to all sources and downstream to all sinks and the total costs can be evaluated.

6 Example

A more advanced example is shown in Figure 3. The user graph in (a) specifies part of a video conference application. The idea is to insert the locally taken picture into the picture of the counterpart. The picture-in-picture multiplexer blends two video streams, one with a small resolution, the other with a high resolution; in this application, the video data generated by a camera (the local picture) and the video data, which is received compressed over a network connection (the picture of the counterpart). The multiplexer can handle raw video data in RGB or YUV format. A filter node, blending a certain logo like the current clock, and a display node as sink complete the user graph. Both nodes can only process RGB video data. Note, that the user graph does not have to include any converters or format specific details.

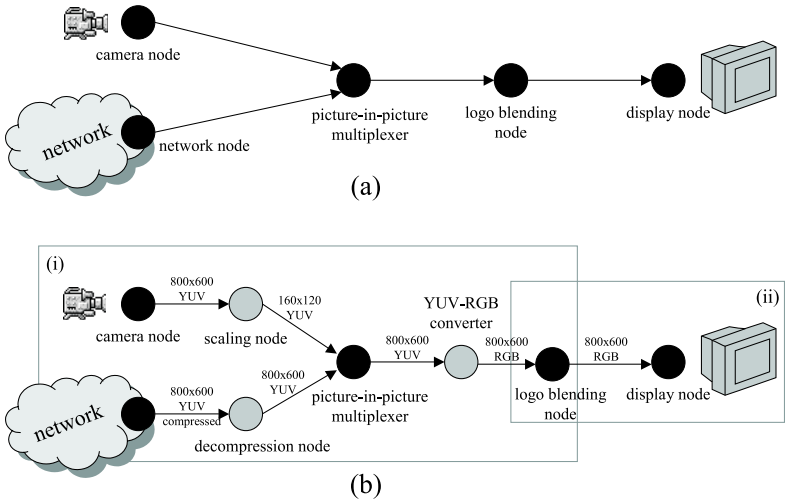

Fig. 3. User graph and result flowgraph of a video conference application.

Figure 3(b) shows the final result of the format negotiation: A converter for decompression has been inserted to fulfill the format constraints of the multiplexer, which can only handle uncompressed video data. The other input-stream of the multiplexer is scaled by an additional converter, since the second input of the multiplexer can only handle small resolution pictures. The color space converter from YUV to RGB is also inserted for connecting the specified logo node. All these steps are performed in one subgraph, denoted in Figure 3(b) by (i). Afterwards, the subgraph (ii) is resolved. As the given formats match, there is no need to insert a converter.

It is noticeable that the YUV to RGB converter is inserted behind the multiplexer. Another solution would have been to convert the video data of both streams individually before they are merged. Since this possibility provides the same quality but needs two relatively expensive conversions instead of only one, this solution was not generated.

7 Implementation

The format negotiation has been implemented in C++ within the Network-Integrated Multimedia framework [15] for Linux. The modified Dijkstra's algorithm uses a STL priority queue to achieve best results. Even with the current non-optimized implementation, the running time of the initialization, format negotiation and construction of a multimedia flowgraph with a total of 20 formats, 15 nodes and 24 edges in the negotiation graph takes only 40ms. A more advanced example with 50 nodes and 100 edges takes 60ms on a standard PC (P-III, 800 MHz) under Linux.

8 Results and Discussion

Given the flexible and quickly changing environment of distributed multimedia applications, we considered the problem of defining a valid flowgraph while minimizing the necessary input by a user. In our scheme the user only needs to specify an incomplete "user graph" containing the key components such as the media sources and specific processing components. The format negotiation then automatically selects additional components that are necessary for a consistent flowgraph which provides the highest "quality" in terms of a weighting function. Additionally, the total costs are considered if more than one solution exists. The resulting components are then connected into a complete flowgraph that can directly be used for media processing. This scheme avoids inflexible template-based approaches and allows to immediate integration of new devices with new format requirements.

We consider this format negotiation an essential part of a distributed multimedia middleware. We plan on extending the basic technique presented here to include cost driven format and QoS negotiation as well as resource allocation (in particular for network bandwidth). Furthermore, further work is required on the fast reconfiguration of flowgraphs given dynamic changes in stream requirements and the environment.

References

1. Microsoft: DirectShow Architecture. http://msdn.microsoft.com/ (2000)
2. Apple: Quicktime 5 API Reference. http://developer.apple.com/quicktime/ (2001)
3. Creek, P., Moccia, D.: Digital Media Programming Guide. Silicon Graphics (1996)
4. Khronos Group: OpenML White Paper V2.0. http://www.khronos.org/ (2001)
5. Sun: Java Media Framework 2.0 API Guide. http://java.sun.com/products/java-media/jmf/ (1999)
6. HAVi: Specification of the home audio/video interoperability (havi) architecture, version 1.1. http://www.havi.org/techinfo/ (2000)
7. Sun: Jini Specifications v1.1. http://www.sun.com/jini/ (2000)
8. Microsoft: Universal Plug and Play Architecture. http://www.upnp.org/ (2000)
9. Hewlett-Packard: JetSend Technology. http://www.jetsend.hp.com/ (2000)
10. Guedes, L., Oliveira, P., Faina, L., Cardozo, E.: QoS Agency: An Agent-based Architecture for Supporting Quality of Service in Distributed Multimedia Systems. In: IEEE Conference on Protocols for Multimedia Systems - Multimedia Networking. (1997)
11. Hafid, A., Bochmann, G., Kerherve, B.: A Quality of Service Negotiation Procedure for Distributed Multimedia Presentational Applications. In: Fifth international IFIP Symposium of High Performance of Distributed Processing. (1996)
12. Rothermel, K., Dermler, G., Fiederer, W.: QoS Negotiation and Resource Reservation for Distributed Multimedia Applications. In: Proceedings of the 1997 International Conference on Multimedia Computing and Systems (ICMCS '97). (1997)
13. Cormen, T., Leiserson, C., Rivest, R.: Introductions to Algorithms. MIT Press (1997)
14. Wayne, K.: Theory of Algorithms. Technical report, Princeton University (2001)
15. Computer Graphics Lab, Saarland University: Network-integrated Multimedia Framework. http://graphics.cs.uni-sb.de/NMM/ (2001)