CHAPTER 5
ADAPTIVE PRESENTATIONS

The proposed architectural framework which supports dynamic media adaptation and an algorithm for constructing an adaptation tree are presented in this chapter. The emerging MPEG-21 standard also proposes a scheme for effective content adaptation. The proposed content adaptation scheme has been designed keeping in mind the need for it to be compatible with the MPEG-21 specifications. Here, the task of choosing a set of services is dealt with efficiently, using the proposed adaptation tree algorithm. The proposed approach can handle multiple requests with a better response time.

5.1 EXISTING WORK

Content adaptation can be broadly classified into static adaptation and dynamic adaptation. In static adaptation, content is created and stored in different formats, specific to each device, by the content provider. In dynamic adaptation, the system gathers information about the device, network and the client, and adapts the content accordingly. Dynamic adaptation can further be classified into three categories: server-based, client-based and proxy-based. The adaptation technique, when done at the server side (or client side) itself, without involving a third party to do the necessary transformations is called server based (or client based) adaptation. The proxy based adaptation consists of employing a third entity between the server(s) and the client(s) to retrieve the client request and perform necessary adaptation on the content received from the server. The generated content is then sent to the client. Here, all
the proxy tasks are designed to behave transparently to client and content servers. *Universal Multimedia Access* (UMA) refers to the ability of any type of terminal to access and consume a rich set of multimedia content. Working towards this goal of universal accessibility, techniques for scalable coding and transcoding that proved to be less computation-intensive were developed by Vetro and Timmerer (2005). In such a case where no limitations exist on the terminal or access network, real-time adaptation becomes a necessity. Some of the existing approaches are briefly presented here.

- **Finite Automata**: A formal framework for the dynamic adaptation of a multimedia document to a given context, described in terms of available resources (i.e., the device, user profile and environment), is presented by Bertolotti et al (2006). The multimedia document is described by means of an automaton that records, at each step, the set of active media and the occurred events. Undeliverable fragments of a document are characterized and semantically equivalent presentation fragments are defined as alternatives to undeliverable ones. In the absence of equivalence, undeliverable media are replaced with those that minimize the loss of information/quality in the presentation. Media items in the fragment are completely presented within the frame, and their removal does not leave unaccomplished media presentations in the rest of the document. This process of adaptation is done in two phases: the first phase checks the feasibility of the presentation and the second phase replaces undeliverable media with deliverable media when mismatch occurs between the input and the required output.

- **Decision Engine**: A decision engine approach to perform the adaptation is presented by Lum and Lau (2002). Content negotiation is performed in 2 steps: preprocessing and real-time processing.

  - **Preprocessing** - Depending on the user’s preference, data type and
required QoS, some scores are assigned to each content type and represented in the form of nodes, called score nodes. These score nodes are generic and can be used with any content. The decision engine processes these nodes and stores them.

- Real-Time processing - Based on the network parameters, device capabilities and content metadata, the decision engine invokes a negotiation algorithm which returns the optimal path score node. The algorithm traverses each node and makes a binary decision. The decision function negotiates iteratively with the score node data structure until it finds a satisfactory score node with a true decision. The choice of the negotiation algorithm depends on the data structure used to store the score nodes. During the pre-processing, the decision engine performs a search on the score nodes and processes the suitable nodes into a suitable data-structure. During the real-time processing, a heuristic search is made on the decision tree using an appropriate algorithm. This makes the searching process more complex.

- Graph Based approach: To transform multimedia content according to the client’s request, satisfying constraints of the device and network profile, a graph based method was proposed by Berhe et al (2005). A transformation prescript provides a list of transformations needed to deliver data, which when replaced by appropriate services available, results in an adaptation graph. When the adaptation graph is being constructed, the initial graph may not be complete. The graph is completed by adding neutral nodes, without changing the semantics of the graph. They are added when the output from one level does not match the input of the next level. If two nodes cannot be connected using any number of neutral nodes, they are removed from the graph.
The final graph may contain many paths from the start node to the end node, but only the shortest path, which gives the execution plan, is chosen. The probability of using the neutral nodes introduced for the completion of the graph is high, since not all the services rendered by one level are compatible with the services rendered by the next level. But the probability of occurrence in the shortest path is very low. This is because more nodes are used in between two levels of the graph, which tends to increase the length of the path. The difference between these two probabilities is wide enough to incur an overhead. The overheads in the existing approaches lead to poor response time to content adaptation.

Illustration:

The proposed approach is a variation of the graph-based approach provided by Berhe et al (2005). The illustration presented by Berhe et al (2005) is briefly given here in order to easily understand the processes involved in the adaptation graph approach. The authors assume the original content to be an English text and the user prefers the data to be in French speech. In order to convert English text to French speech, three basic transformations are involved. Let the transformation prescript be the set $T=\{T_1, T_2, T_3\}$ as shown in Figure 5.1. First, text in English is

![Figure 5.1: Transformation prescript](image)

summarized ($T_1$). It is then translated into French ($T_2$) and is finally converted to French speech ($T_3$). The adaptation graph is constructed as follows. The three elements of the prescript $T_1$, $T_2$ and $T_3$ represent
the available services. For instance, if $T_2$ represents English to French translation, then the set given below (for $T_2$) indicates that there are three ways to convert English to French using the services $s_{21}$, $s_{22}$ and $s_{23}$. Hence, the three nodes of the prescript $T_1$, $T_2$ and $T_3$ are represented by the set of services as given as follows:

\[
T_1 : \{s_{11}, s_{12}\} \\
T_2 : \{s_{21}, s_{22}, s_{23}\} \\
T_3 : \{s_{31}, s_{32}, s_{33}\}
\]

By assuming the quality factors for the input and output of each service as $Q_{in}$ and $Q_{out}$, if the input quality of one service matches the output quality of another service, the nodes are connected to form the adaptation graph as shown in Figure 5.2. If some of the nodes are not connected, neutral nodes are used. In Figure 5.2, since the output of $s_{21}$ does not match with any node in the next level, the neutral nodes $s_{n1}$ and $s_{n2}$

Figure 5.2: A sample unfinished adaptation graph
are used, thus connecting $s_{21}$ to $s_{31}$ indirectly, i.e. if the output of $s_{21}$ is English text (large) and input of $s_{31}$ is French speech. Here, there is no neutral node to connect to $s_{33}$ and hence it is removed from the graph. Effective paths from the input to the output are thus established. The final graph is shown in Figure 5.3. The optimal path is chosen using the

![Final adaptation graph](image)

**Figure 5.3: Final adaptation graph**

quality criteria. The major considerations for quality are the cost and time. The quality criteria of a service $S$ for an operator $t$ is given by:

$$Q(S) = (Cost_{s,t}, Time_{s,t})$$  \hspace{1cm} (5.1)

The user’s preferences of quality and requirements for faster services and lower costs are incorporated while calculating quality value. Let $Q(s_{ij})$ be the quality value criteria of the path and $W_j$ be the weight of each criterion. Then, for each path $P(i)$,

$$Score(P_i) = \sum_{j=1}^{n} Q(s_{ij}) \times W_j$$  \hspace{1cm} (5.2)
where \( n \) is the number of quality criteria taken into consideration. The adaptation planner chooses the path with the maximum value of score, and this searching of optimal path is done through exhaustive computation of the adaptation graph. In this case, the number of transformations required is very limited.

5.1.1 Motivation

The proposed architectural framework involves the use of a proxy server at the client end to process requests from clients, and a proxy server at the server end to perform content adaptation. The proxy at the client end also plays a role in the content adaptation, by generating the adaptation plan in the form of an adaptation tree for each client request, taking into consideration the device and user profiles. This framework can also be modeled using MPEG-21 standard (Vetro and Timmerer 2005), with a in-built layer that performs the core adaptation function. A new graph-based content adaptation algorithm that helps to effectively generate the adaptation tree has been proposed by Berhe et al (2005). This work is a variation of the algorithm proposed for content adaptation. The proposed algorithm runs in linear time, making it more efficient than the algorithm put forth by Berhe et al (2005).

5.2 PROPOSED ARCHITECTURE

A proxy based framework, which involves the use of proxies at both client end and server end, has been proposed, and its architecture is depicted in Figure 5.4. Client requests are forwarded to and processed by the client proxy. The client proxy then extracts the client profile from its database and bundles it with the device profile, which is automatically received along with the client request, to create a destination profile. The request is integrated with the
destination profile which is forwarded to the server proxy. The server proxy retrieves the content profile of the requested presentation from the content server and creates a source profile from it. The server proxy checks if the content profile requires any adaptation. If adaptation is required, the server proxy makes a list of available adaptation services, where the availability of a service is based on its queue length. The server proxy sends this list and the source profile to the client proxy. The client proxy uses the source profile and the destination profile to construct an adaptation tree. The final tree is sent to the server proxy. The server proxy processes the content by implementing the tree and sends the adapted media along with its presentation schedule back to the client proxy.

To ensure that the adaptation services requested by the clients will be available till the processing of the requests are over, a FIFO queue of request for the service is maintained at the server proxy for each and every service, referred to as service queues. Each service queue has an elastic limit based on the flow of traffic at that instant. This elastic limit is always less than the maximum length of the queue. Once the queue reaches its elastic limit, the service is no longer shown as available for further requests, but clients who had requested access for the service before it reached its elastic limit will be put in the service queue till their requests are completed. The function of each component in the framework is described as follows:

- **Client**: A client is an entity in the network that requests for information. This request is integrated with the device profile and is forwarded to the client proxy.

- **Client Proxy**: When the client proxy receives the request for a presentation, it retrieves the client profile from its database and creates a destination profile for this request. Then, it establishes a connection with the server proxy, and receives the list of adaptation services available
Figure 5.4: Architecture for content adaptation

and the source profile, if adaptation is required. The adaptation tree is constructed by the client proxy and sent to the server proxy. If adaptation is not required, the client proxy receives instead the content and presentation schedule which is forwarded to the client for playout.

- **Server Proxy:** The server proxy generates the source profile which is a combination of the content profile and list of adaptation services. The server proxy also keeps track of the queues associated with every adaptation service. When the server proxy receives the adaptation tree from the client proxy, it processes the media, adapting it according to the tree. The processed media is sent back to the client proxy along with the presentation schedule as well as the media that are part of the presentation but did not need adaptation.

- **Content Server:** The content server maintains a database of presentations and provides the content profiles to the server proxy when requested for.
Figure 5.5: Flow of requests of data

The construction of the adaptation tree takes place at the client side and the actual adaptation process takes place at the server side. This balances the content adaptation service load on either proxies.

Content adaptation using MPEG21:

The flow of data from the client to the server and back to the client are performed in various steps as shown in Figure 5.5.
(a) As a first step, the client’s request along with the device profile is sent to the client proxy.

(b) The client proxy bundles the received device profile along with the client profile from its database to create the destination profile, and sends it to the server proxy.

(c) The destination profile is sent to the server proxy.

(d) The server proxy requests the content servers for the content profile. The content servers maintain a list of DIDs associated with every document.

(e) The requested content profile is sent to the server proxy.

(f) The server proxy now creates the source profile.

(g) With the available information, the server proxy processes the request by comparing the source and destination profiles to make a decision on whether adaptation is required or not. If adaptation is required, the list of DIDs are sent to the client proxy.

(h) The list of services in the form of DIDs are sent to the client for acceptance. The client is allowed to choose services from this list and make changes in the parameters, if necessary.

(i) The client sends its acceptance to a list of services.

(j) The adaptation graph is generated by the client proxy as per the user’s request.

(k) This graph is sent to the server proxy for adaptation to take place.

(l) A request is sent to the content server for the DIs.

(m) The requested DIs are sent to the server proxy.

(n) The server proxy uses the standardized functions specified in the DIA to guide the adaptation process based on user interactions.

(o) The adapted output is sent to the client.

Here, the MPEG-21 multimedia framework provides a mechanism to add methods to a DID that allow DIs to be processed in an interoperable way.
In this case, before the generation of the adaptation graph, the DIDs are processed as required by the user. The various DIMs available along with their respective choices of arguments are presented to the user. Based on the user’s selection, the DIMs are executed. This selection process is repeated until the user is satisfied with the DID. The final DID is used in the process of generating the adaptation graph.

5.2.1 Adaptation Tree Algorithm

A tree is constructed, to perform content adaptation. This involves the concurrent generation of a forward path and a backward path. The forward path uses the source profile as its base while the backward one makes use of the destination profile. Various possible content transformations are represented as nodes. The nodes are represented in the following format:

\[ \text{Node}(\text{input}, \text{output}, \text{weight}, \text{parent}, \text{child}) \]  

(5.3)

where input and output represent edges that indicate the media/data flow from one adaptation service to another, weight is a function of the time \(Q_{\text{time}}\) taken for the transformation and the cost \(Q_{\text{cost}}\) incurred during the transformation, i.e

\[ \text{Weight} = f(Q_{\text{time}}, Q_{\text{cost}}) \]  

(5.4)

and parent and child represent the previous and next node of the node in consideration. In Algorithm 5.1, the tree \(T\) is initialized to contain the two dummy nodes Start and End. All the nodes are assumed to be sorted according to their weights in the client proxy. \(N\) is the set of all nodes while input and output represent the media formats. Lines 6-29 of the algorithm are repeated till the complete tree is obtained. Lines 10-13 of the algorithm checks whether a common node linking the forward path with the backward path exists. If
so, the two paths are linked through the common node and a complete tree is obtained. Lines 16-17 append a node to the forward path while lines 20-21 append a node to the backward path, thereby completing the tree. Both these steps ensure that no cycle can occur in either path by keeping track of the nodes already connected to the existing tree. The algorithm returns a tree $T$ consisting of a path from $Start$ to $End$. The total cost ($W$) of this path is computed as:

$$W = \sum_{i=1}^{n} (w_i \ast q_i) \quad (5.5)$$

where $w_i$ is the weight of node $i$ and $q_i$ is the queue length of the service represented by the node $i$ and $n$ is the length of the path. This cost represents the response time, which is communicated to the client as the maximum time the client would need to wait before the presentation commences. If the client is satisfied with this $QoS$ parameter, maximum response time, the client confirms the request which the client proxy forwards along with the adaptation tree to the server proxy. The server proxy implements the adaptation tree and forwards the requested adapted presentation to the client proxy.

The function $void Update(node fromNode, node toNode, node startNode, node endNode)$ provided in the algorithm is used to add the nodes from and to to the existing graph through the edges linking $start$-to and $from$-end. It also changes the values of $ip$ and $op$ in order to proceed with the construction of the tree. The boolean function $SetParent(Node n)$ provided in Algorithm 5.2 is used to assign the parent node of the node $n$ in the forward path. It checks for the existence of loops or longer paths and eliminates them if found. It returns true if the tree is complete, else returns false. Similarly, the boolean function $SetChild(Node n)$ provided in Algorithm 5.3 is used to assign the child node of the node $n$ in the backward path. It checks for the existence of loops or longer paths and eliminates them if found. It returns
Algorithm 5.1: Adaptation_Tree()

Input: N : Set of all nodes,

Output: Tree T

1. Initialize the parent and child of all nodes to NULL;
2. ip ← input service;
3. op ← output service;
4. prev ← start node;
5. next ← end node;
6. while true do
7.    P ← Q ← NULL;
8.    for each n ∈ N do
9.       if n.input = ip and n.output = op then
10.          n.parent ← prev;
11.          n.child ← next;
12.          update(prev, n, n, child);
13.          return T;
14.    end
15.   else if n.input = ip then
16.       P ← P ∪ {n};
17.       if (SetParent(n)) then return T;
18.   end
19.   else if n.output = op then
20.       Q ← Q ∪ {n};
21.       if (SetChild(n)) then return T;
22.   end
23.   end
24. x ← first(P);
25. y ← first(Q);
26. update(prev, x, y, next);
27. prev ← x;
28. next ← y;
29. end

true, if the path is determined, or else returns false.

5.2.2 Illustration

Consider that a client has made a request for a presentation from the server. Assume that the destination profile indicates that the client prefers a summarized speech in French in place of any text. It is assumed that the
Algorithm 5.2: Function: Boolean SetParent(Node n)

1 if prev = n then
2    Remove 'n' from 'P'
3    return false
4 end
5 if n.parent = -1 and n.child ≠ -1 then
6    n.parent ← prev
7    x ← n.child
8    Remove path from 'next' to 'x'
9    update(prev, n, n, x)
10   return true
11 end
12 else if n.parent ≠ -1 then
13    x ← prev
14    prev ← n.parent
15    Remove path from 'prev' to 'x'
16 end
17 else
18    n.parent ← prev
19 end
20 return false

Algorithm 5.3: Function: Boolean SetChild(Node n)

1 if next = n then
2    Remove 'n' from 'Q'
3    return false
4 end
5 if n.child = -1 and n.parent ≠ -1 then
6    n.child ← next
7    x ← n.parent
8    Remove path from 'x' to 'prev'
9    update(x, n, n, next)
10   return true
11 end
12 else if n.child ≠ -1 then
13    x ← next
14    next ← n.child
15    Remove path from 'x' to 'next'
16 end
17 else n.child ← next return false
Figure 5.6: Construction of adaptation tree

requested presentation has one large English text file and hence an adaptation is required. Let $n$ be the number of nodes (adaptation services) available. In the illustration, $n$ is considered to be 8 and the nodes are depicted in Figure 5.7. The eight services depicted in the Figure 5.7 play the roles of either a summarizer, a language translator or a media converter. Initially, the algorithm generates the start node and the end node which do not represent any service. In the illustration, they are Start and End respectively. Since the input is a file containing English text, the algorithm assigns all nodes which
Figure 5.7: Adaptation services

have the same input as the children of node Start. Similarly, all nodes having
the output as a short French speech are assigned as the parents of the node End
(Figure 5.6(a)). From the possible 8 nodes, the algorithm chooses the shortest
path from the start and the end sides of the tree. The input and output values
are changed according to the nodes chosen as successor nodes on either paths
(Figure 5.6(b)). Repeating the procedure with nodes N1 and N7, a tree is
obtained after elimination of recursion as shown in Figure 5.6(c). When the
process is repeated for the third time, a node, N8 is found, on both the paths.
Thus, the forward and backward paths are linked through the common node $N_8$, thereby completing the adaptation graph as shown in Figure 5.6(d). The tree is then generated without having to visit all the nodes.

5.2.3 Proof of Correctness

Let $T$ be the adaptation tree generated. The correctness of the algorithm can be verified if the generated tree is the tree with the minimum cost. Let $T_1$ be a tree with minimum cost path. If $T_1 = T$, then the algorithm is proved to be correct. Suppose $T_1 \neq T$ and let $n_1$ be the node in $T_1$ which has a lesser weight than the node $n$ in $T$. Since $T_1$ is the tree with minimum cost, it has a path joining the start node to the end node. As the path is traversed, an edge joining the node $n_1$ to the tree either through the forward path or through the backward path must be encountered. At every iteration of the adaptation algorithm, at least one node is added to the tree. Now, during an iteration, when $n_1$ was added to $T_1$, $n$ could also have been added if its weight were less than $n_1$. This is because $n$ and $n_1$ are adjacent nodes and the algorithm keeps track of all adjacent nodes. Since $n$ was not added, it is concluded that

$$weight(n) \geq weight(n_1)$$ (5.6)

The algorithm chooses the first node in each set to add to the tree at the end of each iteration. Since the nodes are assumed to be sorted according to their weights prior to the execution of the algorithm, the first node of a set is the node with the least weight. So, the node $n$ is added to the tree only if

$$weight(n) = weight(n_1)$$ (5.7)

Thus, it is concluded that the total weight of the tree $T$ is equal to the total weight of the tree $T_1$. Hence, it can be concluded that $T$ is also a tree with minimum path cost, thereby proving the correctness of the algorithm.
5.2.4 Complexity Analysis

Consider the Algorithm Adaptation-Tree. Let $n$ be the total number of nodes available at the server. Let $m$ be the number of media transformations required. The SetParent, SetChild and Update functions can be executed in constant time. The algorithm has $n$ SetParent operations and $n$ SetChild operations require $O(n)$ time. The algorithm also has $m$ Update operations that require $O(m)$ time. Hence, the time complexity of Algorithm is $O(mn)$.

5.3 SIMULATION RESULT

In a distributed environment, the traffic and the number of service requests varies from time to time. This variation has been taken into account in the proposed approach. The proposed architecture places an upper bound on the delay caused due to network traffic. Every adaptation service has a fixed service rate which is assumed to be equal for all services present. Assuming that there are $r$ requested services and $c$ clients requesting them, then the delay($D$) would be maximum when all $c$ clients request for the same services and minimum when all $c$ clients request for different services. Since each service has a queue whose maximum length is fixed, the service will become unavailable if the number of clients requesting for that service reaches the upper bound of the queue. When this happens, the service becomes unavailable for all clients requesting this service in future.

From Figure 5.8 it can be seen that if the number of clients $c$ are fixed, then the delay $D$ for each client increases as the number of requests $r$ increases. Similarly from Figure 5.9, it can be seen that by fixing the number of requests and if the number of clients increases, the delay for each client decreases. Both the graphs have been plotted in the case where service time
for each service is fixed at $t = 30$. Thus, it is concluded as the delay incurred in completion of the clients request depends upon the number of clients requesting for the same service and the number of services requested by the clients.

**Figure 5.8: Service requests vs delay**

**Figure 5.9: Number of clients vs delay**

### 5.4 SUMMARY

In this chapter, a content adaptation mechanism built on a proxy-based architecture has been presented. Proxy-based approaches help to reduce the load on the content servers and precisely direct requests to the appropriate set of adaptation services. Thus when the client requests for a presentation,
the content in the presentation is adapted to suit the device capabilities and client preferences. A database of services supporting conversion of content from several types of input forms, generating several types of output forms, is also maintained by the system. The system needs to dynamically decide which set of services to employ in order to quickly get the content into its final form. An adaptation tree generated at the client proxy for each client request helps to choose the set of services that would provide the best response time. The proposed algorithm generates the tree dynamically, eliminating the cycles as it is generated. It runs in linear time, making it more efficient than the existing approaches. The construction of the adaptation tree takes place at the client proxy, and the actual adaptation process takes place at the server proxy. This balances the content adaptation service load on either proxies. A dynamic media adaptation mechanism for distributed systems, efficient in terms of its response time to client requests, has been presented. In addition to this proxy based approach, an integrated use of MPEG-21 standard that helps to perform adaptation efficiently using the digital item adaptation specification has also been presented.