Structured Scalable Meta-formats (SSM) Version 2.0 for Content Agnostic Digital Item Adaptation - Principles and Complete Syntax

Debargha Mukherjee, Geraldine Kuo, Amir Said
Imaging Systems Laboratory
HP Laboratories Palo Alto
HPL-2003-71
April 7th , 2003*

E-mail: {debargha, gkuo, said} @hpl.hp.com

scalable bit-stream, format-agnostic, content-agnostic, adaptation engine

This paper develops an end-to-end methodology for representation and adaptation of arbitrary scalable content in a fully content non-specific manner. Scalable bit-streams are naturally organized in a symmetric multi-dimensional logical structure, and any adaptation is essentially a downward manipulation of the structure. Higher logical constructs are defined on top of this multi-tier structure to make the model more generally applicable to a variety of bit-streams involving rich media. The resultant composite model is referred to as the Structured Scalable Meta-format (SSM). Apart from the implicit bit-stream constraints that must be satisfied to make a scalable bit-stream SSM-compliant, two other elements that need to be standardized to build a complete adaptation and delivery infrastructure based on SSM are: (1) a binary or XML description of the structure of the bit-stream resource and how it is to be manipulated to obtain various adapted versions; and (2) a XML specification of outbound constraints derived from capabilities and preferences of receiving terminals. By interpreting the descriptor and the constraints a universal adaptation engine can adapt the content appropriately to suit the specified needs and preferences of recipients, without knowledge of the specifics of the content, its encoding and/or encryption. With universal adaptation engines, different adaptation infrastructures are no longer needed for different types of scalable media.

Table of Contents

Abstract

This paper develops an end-to-end methodology for representation and adaptation of arbitrary scalable content in a fully content non-specific manner. Scalable bit-streams are naturally organized in a symmetric multi-dimensional logical structure, and any adaptation is essentially a downward manipulation of the structure. Higher logical constructs are defined on top of this multi-tier structure to make the model more generally applicable to a variety of bit-streams involving rich media. The resultant composite model is referred to as the Structured Scalable Meta-format (SSM). Apart from the implicit bit-stream constraints that must be satisfied to make a scalable bit-stream SSM-compliant, two other elements that need to be standardized to build a complete adaptation and delivery infrastructure based on SSM are: (1) a binary or XML description of the structure of the bit-stream resource and how it is to be manipulated to obtain various adapted versions; and (2) a XML specification of outbound constraints derived from capabilities and preferences of receiving terminals. By interpreting the descriptor and the constraints a universal adaptation engine can adapt the content appropriately to suit the specified needs and preferences of recipients, without knowledge of the specifics of the content, its encoding and/or encryption. With universal adaptation engines, different adaptation infrastructures are no longer needed for different types of scalable media.

1. Introduction

Users access the Internet today using devices ranging from puny handhelds to powerful workstations, over connections ranging from 56 Kbps modems to high speed 100 Mb/s Ethernet. Even though the available bandwidth, display and processing capabilities may continue to grow following Moore’s law, the heterogeneity and the diversity of capabilities at any point in time is here to stay. On the other hand, as bandwidth and other factors grow, so will the richness of media that would need to be delivered to users. Under these circumstances, a rigid media representation format, producing content only at a fixed resolution and quality is clearly inappropriate. A delivery system based on such a compression scheme can only deliver content satisfactorily to a small subset of users interested in the content. The rest, either does not receive anything at all, or receives poor quality and/or resolution relative to the capabilities of their network connections and /or accessing devices. The inability to cater to this diversity has been a determining factor that stunted growth of new rich media, because static rich content would cater only to power users comprising a small fraction of the whole. The bottom line is, without adequate focus on seamless content adaptation, accessibility and usability of media content will always remain limited.

1.1 Multiple versions

A practical approach to catering to heterogeneity is one where multiple versions of any piece of media, suiting a variety of capabilities and preferences, are maintained simultaneously. While this approach works well with delivery models where the recipient directly connects to a media originator, for any other multi-hop, multi-recipient delivery scenario, there is inevitable redundancy leading to wastage of bandwidth and storage. This is especially so, when the media creator intends to provide a wide range of choices for adaptation catering to a large consumer base, and therefore needs to maintain a large number of versions differing in a variety of ways. In other words, this approach does not scale well with the amount of flexibility a media creator would like to provide.

Note however, that since the multi-version case is a fully redundant special case of true scalability to be described next, the framework proposed here still applies. This is also true for hybrid bit-streams that combine multiple versions with true scalability. In all cases, we still need protocols to describe content adaptation choices and request adaptations in a flexible way that from the receiving terminal.

1.2 Scalable Bit-streams

In order to provide a solution more elegant than maintaining multiple versions to cater to diversity, scalable compression formats have been proposed. In a scalable bit-stream, smaller subsets of the whole produce representations at lower resolution, quality, etc. Different subset bit-streams extracted from the full parent bit-stream, can readily accommodate a variety of users by automatically maximizing multimedia experience for a given user's computing power, connection bandwidth, and so on. By adapting rich media content written for high-end machines to less powerful machines in various ways, the overheads involved in producing different versions for different scenarios can be virtually eliminated. Furthermore, content created today at the highest possible quality, remains 'timeless' when represented in a scalable format, and the experience it provides gradually increases, as the power of machines, connection speeds, etc. improve.

There are various types of bit-stream scalability that can be designed, depending on the type of media. For example, *SNR* (quality) scalability refers to progressively increasing quality as more and more of the bit-stream is included, and applies to most types of media. *Resolution* scalability refers to fineness of spatial data sampling, and applies to visual media such as images, video, 3D etc. *Temporal* scalability refers to fineness of sampling in the time-domain, and applies to video and other image sequences. There are several types of scalability pertaining to audio, such as number of channels, width of the frequency band. In the future, with the evolution of newer, richer and more interactive types of media, there will be newer types of scalability, for e.g. different kinds of interactivity scalability, which we do not even know yet.

In recent years, there has been a great deal of interest in the research community on scalable compression of various types of digital media. Here the challenge is to obtain a scalable representation without sacrificing compression efficiency. So far however, it is only in the area of still image compression that it has been possible to obtain efficient scalable coders that even improve compression performance (Ex. EBCOT [4], SPIHT [5], VSPIHT [6], EZW [7]). EBCOT [4] led to the evolution of the new JPEG2000 [8] standard for contone images. The JBIG and JBIG-2 standards for binary images are also scalable. Besides images, there has been considerable effort to obtain efficient and compact scalable representations of video, audio, and other types of media. In fact, most existing media encoding standards today, [9], [10], [11], [12], [13], [14] incorporate various scalability modes, although generally there is a loss in compression efficiency to use them, and they are not fully scalable. It is only recently that new fully scalable video codecs, such as 3D-ESCOT [15] and MC-EZBC [16], [17], [18], have been shown to be viable. Fully scalable video coding is currently under exploration in MPEG [19], [20], [21], [22]. Also, there is ongoing activity in MPEG-21 DIA related to delivery of scalable bit-streams [23], [24], [25], [26], [27]. With evolution of new types of media, it is conceivable that there will be emphasis on scalable representations for them as well, although not every type of content can be standardized.

A scalable bit-stream does not always have a single type of scalability. In fact, different types of scalability may co-exist in a multi-dimensional structure, so as to provide a wide range of adaptation choices. For example, while SPIHT [5], its predecessor EZW [7], and several of their derivatives [6] only support SNR scalability, EBCOT [4] endeavors to combine quality scalability and resolution scalability in a common format, to enable distribution and viewing over a wider variety of connections and devices.

Furthermore, in new rich media, different media elements are often clubbed together to provide a composite media experience. For example, an image with audio annotation and some animation provides a composite experience of a presentation using three elemental media elements (an image, an audio clip, some animation data). The composite rich media leads to newer types of scalability specific to the media, because certain non-critical elements may be dropped to accommodate other more critical ones within the limited resources of the network and a recipient.

1.3 Scalable Content Adaptation and Delivery Infrastructures

In order to unlock the full potential of a scalable bit-stream, the format alone is insufficient. It is necessary to develop and deploy complete infrastructures that support appropriate adaptation and delivery of such content, so that a diverse recipient base can experience it with a seamless ease of use.

For example, even though the JPEG2000 [8] format itself is very powerful, the lack of a complete infrastructure that supports appropriate transcoding of JPEG2000 content and delivery to a heterogeneous recipient base has severely restricted the usability of its scalable features. In recent years, a great deal of attention has been focused on delivering streaming video over the Internet or wireless [28], [29], [30]. In order to reach heterogeneous recipients in a dynamic transmission environment, video standards of MPEG-X (mostly MPEG-4) [9], [10], [11], [12] and H.26X [13], [14] families incorporate various forms of scalability. Although rudimentary in scope, functionality and efficiency, as compared to JPEG2000, they hold considerable promise for supporting diversity. Nevertheless, scalable video over the Internet has been limited to maintaining multiple versions for a few different types of connections, because complete infrastructures that support transcoding and transport of scalable video formats are non-existent. It is very recently that a new standardization effort has started in MPEG on fully scalable video coding. However, without adequate focus on content adaptation in the network, its scalability features would remain unexploited.

1.4 Need for Media-type agnostic Adaptation Infrastructures

Any infrastructure, is expensive to deploy, and requires significant financial commitments from patron companies or patron consortia. Under these circumstances, use of a standardized format for the content is desirable in order to guarantee constancy. On the other hand, standards evolve all the time. There are often extensions added to existing standards to support better and enhanced features, and it is conceivable that as new types of media beyond traditional images, video and audio evolve it would be necessary to create new standards for their representation. It can also be argued that since standards take several years to come into effect, typically much longer than is commensurate with the normal pace of change in the multimedia industry, it would become more and more difficult to expect standards to support the representation and delivery of new types of content.

Even if compact scalable formats evolve for every new type of media, the inevitable difference in the structure of the content would necessitate use of different infrastructures or components thereof for scalable delivery of different types of media. The expenses involved present a very formidable obstacle in adoption of such new media and supportability of its scalability features.

The only way out is to develop infrastructures for content adaptation that are *media-type agnostic*. Such universal adaptation infrastructures only need to be deployed once to support adaptation and delivery of all types of scalable media, as long as they conform to certain loose restrictions on the encoding structure. Use of universal infrastructures that support delivery and transcoding of a wide variety of media types in a convenient manner is the key to successful adoption of new scalable media.

1.5 Security

The need for secure communication of media content is already being felt, and in the future will be the order of the day. In order of guarantee full end-to-end security, it will be necessary to use delivery architectures where no codec-specific elements are used in the entire path from the content server and perhaps including it, to the receiving terminal. Anywhere in the network that a codec-specific element is used, is potentially a security breach point.

Even in such a secure transmission scenario, midstream content adaptation to cater to diversity would be a necessity. Secure end-to-end streaming using scalable packets has been demonstrated earlier [28], [29]. However, to enable secure content adaptation in a content-agnostic manner, it is necessary to empower network adaptation engines to make decisions about possible adaptations, even when they do not understand the semantics of their decision. The only way to do this is to provide information needed to make decisions in a compact way using a mathematical abstraction that is content-agnostic, and incorporate them into generic descriptors that an adaptation engine can process.

1.6 Motivation for this work

In order to enable universal *media-type agnostic adaptation* infrastructures for scalable media, we propose a framework for scalable media representation that formalizes a loosely defined model or meta-format for all scalable media types rather than a single format for a specific media type (such as JPEG2000 for images). The model is called **S**tructured **S**calable **M**eta-format (**SSM**). Although the meta-format needs to be standardized, it operates at a more abstract level than traditional standards, and requires only format compliance in a loose manner. All compressed scalable bit-streams that need be adapted using the universal adaptation infrastructure must be *SSM-compliant, i.e.* must conform to the meta-format. Besides the bit-stream constraints imposed by SSM-compliance, in order to enable end-to-end adaptation and delivery of SSM compliant media, it is also necessary to standardize the languages for conveyance of information, both between an adaptation engine and the media server, as well as between the

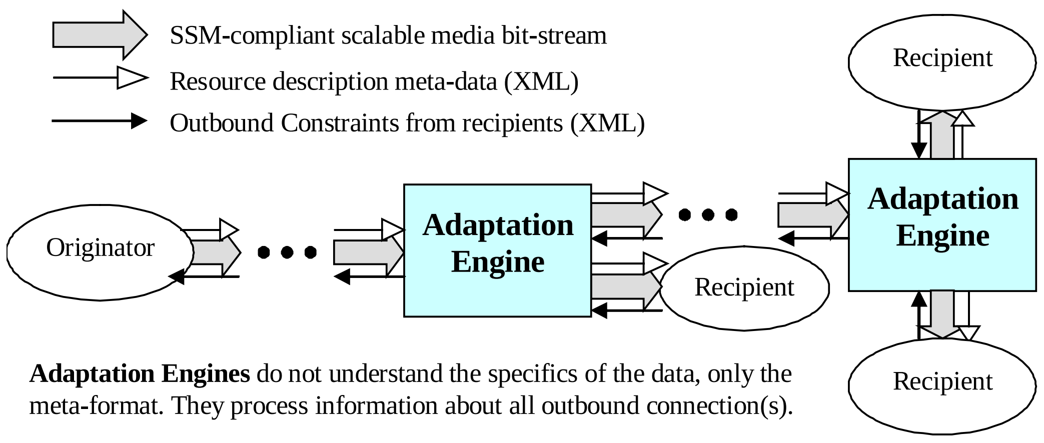

SSM-compliant scalable media bit-stream
Resource description meta-data (XML)
Outbound Constraints from recipients (XML)
Recipient
Originator
Adaptation Engine
Adaptation Engine
Recipient
Recipient
Adaptation Engines do not understand the specifics of the data, only the meta-format. They process information about all outbound connection(s).

Figure 1. Media-Type-Independent Adaptation and Delivery Chain

adaptation engine and the media receiver. Adaptation engines in the delivery chain need to be able to interpret this communication for adaptation purposes.

Media adaptation and delivery infrastructures based on SSM and standardized communication with adaptation engines would be truly media-type- and content-independent, in that they can adapt different types of content, both that are currently available (images, video, audio) as well as those that would evolve in the future (different types of new 3D media, composite media etc), in a secure manner, as long as they comply with the model.

2. Digital Item Adaptation with SSM

2.1 SSM based Delivery Model

Consider Figure 1, which shows a generic media delivery model, where media data created by the originator is routed through an arbitrarily long chain of adaptation engines before reaching an eventual recipient. It is assumed that both the originator of the media as well as the software or hardware system used to experience it at the recipient end understand the actual media-encoding format. It is likely that either the same company created both the media content and the experiencing system, or the creator opened up its technology for vendors to develop the experiencing system, or the media format is an SSM-compliant open standard.

Irrespective of the actual content-type and its encoding however, the scalable resource bit-stream is conformant with SSM, which all intermediate adaptation engines can interpret and manipulate. These engines receive SSM compliant scalable content, and deliver adapted content over multiple outbound streams. All content after adaptation is also SSM meta-format compliant so that it can be re-adapted at a subsequent stage of delivery.

Along with the SSM-compliant media bit-stream an adaptation engine also processes a description meta-data (shown in thin white arrows in Figure 1) that contains vital information for the adaptation engine about all possible adaptations. The SSM model restricts the possible adaptation choices and allows a compact representation of this description. The adaptation engine not only adapts the media bit-stream but also the description meta-data, so that a subsequent stage of adaptation can be applied. There are a

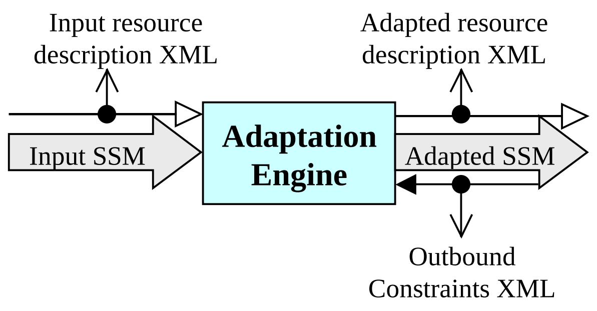

Input resource description XML
Adapted resource description XML
Adaptation Engine
Input SSM
Adapted SSM
Outbound Constraints XML

Figure 2. Adaptation Engine external model

variety of possibilities for representing and conveying this information to adaptation engines. While in MPEG-21 DIA the trend is to use XML, it is to be noted that representing this information in binary form as part of the media bit-stream itself is a straightforward extension, and may even be preferred based on considerations of compactness and manageability. This information is referred to as *resource description metadata*.

It is also assumed that each adaptation engine has knowledge of the aggregated capabilities and preferences of all eventual recipients connected to each of its outbound streams. This information mostly originates from the recipients (shown in thin black arrows in Figure 1), but parts may be sensed by transcoders themselves, as it is aggregated up the adaptation chain by the delivery infrastructure involved. For a particular engine for a particular outbound connection at adaptation time, this information is referred to as its *outbound constraints*, which in general may change dynamically.

Note that while the originator/creator of the media as well as the recipients/consumers of the media must have specific knowledge about the encoding in order to provide an experience for the end-user, the intermediate infrastructure does not need to know what the content is and how it has been encoded in order to adapt appropriately. The adaptation operation is based purely on an interpretation of the resource descriptor metadata and the outbound constraints, and does not depend on the specifics of the actual content. Furthermore, the content itself can be encrypted, and transcoding can still proceed as before in the encrypted domain.

While adaptation engines in Figure 1 are solely functional blocks, in reality they can be part of media servers from where offline or online content originates; midstream routing servers through which scalable content is transcoded and routed; or edge servers that connect directly to eventual recipients. Also, the generic delivery model considered can collapse to as simple as a client-server delivery system where a client requests content from a media server with specified capabilities and preferences, and gets appropriately adapted content directly from it. In this case, the functional adaptation engine would be part of the media server itself.

2.2 Isolated Transcoder Model

In order to understand the scope of the technology, we isolate a single input single output functional transcoder from the end-to-end delivery model, and show its external model in Figure 2. As discussed above, the adaptation engine receives a SSM compliant piece of scalable media, which it must adapt appropriately and forward in a SSM compliant manner to an eventual consumer or another adaptation engine. Along with the media bit-stream it also receives a *media description* XML providing the specifics of how the bit-stream is to be adapted for various adaptation options, as well as an *outbound constraints XML* providing a specification of the capabilities and preferences of its output connection. Based on the information contained in the two XMLs, an adaptation engine makes certain adaptation decisions, performs the adaptation operation based on the decisions to the input SSM-compliant stream to deliver SSM-compliant adapted content to its outbound connection, and updates the media description XML for use in a subsequent adaptation stage.

The internal model for the adaptation engine is shown in Figure 3. In particular, it consists of the following functional blocks: 1) a parser for the resource description meta-data; 2) a parser for the outbound constraints specification, 3) an optimizer to decide on transcoding options, 4) a SSM resource adaptation engine to adapt the resource based on adaptation decisions made by the optimizer, and 5) a resource description metadata adaptation engine to modify the resource description based on decisions made.

It is to be noted that the SSM framework does not determine the operation of the optimizer module in the adaptation engine. The way the adaptation engine arrives at the optimal adaptation decisions based on the resource description and outbound constraints XMLs is open to implementation. However, everything else in the adaptation engine is more or less determined by the proposed adaptation framework. Schemas and other semantics to be described later standardize the two XMLs used. Furthermore, once the decisions have been made, the resource and description adaptation is also deterministic.

2.3 Relation to Network Packetization

It is important to realize that while SSM is about formats for generic scalable content and meta-data describing how format-compliant scalable content is to be adapted, in an actual delivery scenario the content would probably need to be packetized and transmitted. In this regard, among various design choices, there are two that are of particular interest, one based on interpretation of SSM as a file-format, and another based on interpretation as a packet-format.

In the file-format usage case, the scalable resource is actually much larger than a typical network packet. Either the adaptation engine adapts an entire SSM file in one shot

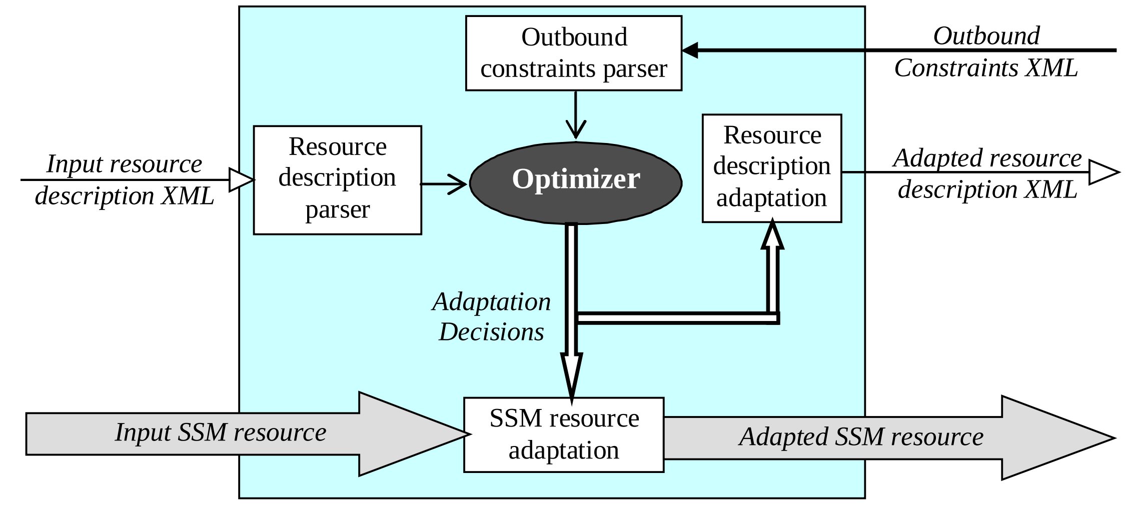

Outbound constraints parser
Outbound Constraints XML
Input resource description XML
Resource description parser
Optimizer
Resource description adaptation
Adapted resource description XML
Adaptation Decisions
Input SSM resource
SSM resource adaptation
Adapted SSM resource

Figure 3. Adaptation Engine Internal model

before network packetization and transmission, or the adaptation is conducted down-stream possibly in multiple stages. In the latter case however, it is important to realize that it is not necessary that the entire SSM compliant resource be available at the adaptation engine before the adaptation operation can commence. In fact, the resource description and the outbound constraint specifications are all that are needed for an adaptation engine to decide how to adapt the media content. As long as the meta-data has been received in full, the scalable bit-stream resource in Figure 3 may come in stages in multiple network packets, and either forwarded or dropped or partially dropped by the engine as they arrive, based on the adaptation decisions already made. Thus, the same adaptation model applies both to files transcoded in one shot as well as to a streamed file.

In the packet-format case, the entire SSM compliant content comprises one packet, which can be adapted by a mid-stream adaptation engine and transmitted. Packet based scalable adaptation based on truncation has been considered before [28], [29]. In this scenario, it may make more sense to include the resource description as part of the packet, using a form of binary encoding rather than XML.

In the rest of this document, we will describe the specifics of the SSM framework: how a generic scalable bit-stream is modeled and what the required constraints are, what information is contained in the resource description metadata (XML) that goes with the resource, and how the capabilities and preferences are conveyed in the outbound constraints specifications (XML). We will also describe how the adaptation decisions are made at a network adaptation engine.

3. Modeling scalable bit-streams

A scalable bit-stream is one where smaller subsets of the whole produce representations at lower quality, resolution etc. Different types of scalability (e.g. SNR, Resolution, Temporal, Interactivity, etc.) apply to different types of media, and often more than one kind is combined. Furthermore, in rich media content several independent elements can be combined, (e.g. video, audio). From an understanding of how a generic scalable bit-stream is naturally organized, we propose a logical model for all scalable bit-streams, referred to as the SSM model.

3.1 Symmetric Nested Scalability Structure

The proposed SSM framework is based on the assumption that any scalable bit-stream component inherently contains logical nested tiers of scalability based on possible adaptations that can be performed on the bit-stream, as shown in Figure 4. Using zero-based indexing, the bit-stream is first divided logically into multiple layers of tier 0 scalability. Here tier 0 is an abstraction, and depending on the actual content it may mean any one of resolution, temporal, SNR and so on. Data segments in tier 0 layers, are further divided into layers of tier 1 scalability, and so on. Again, tier 1 is an abstraction, and may mean different things based on the actual media content. And so on. In addition, it is to be noted that in most useful scalable bit-streams the scalability structure is symmetric. That is, the number of tier 1 layers in each tier 0 layer is the same, and so on.

As an example, consider a JPEG2000 bit-stream, which can be readily cast into this logical-bit-stream-format. In one of the scalability progression modes in JPEG2000 – RLCP – the highest tier is resolution scalability, and within the resolution scalable layers there are nested SNR scalable layers, followed by color layers and precinct layers. In an

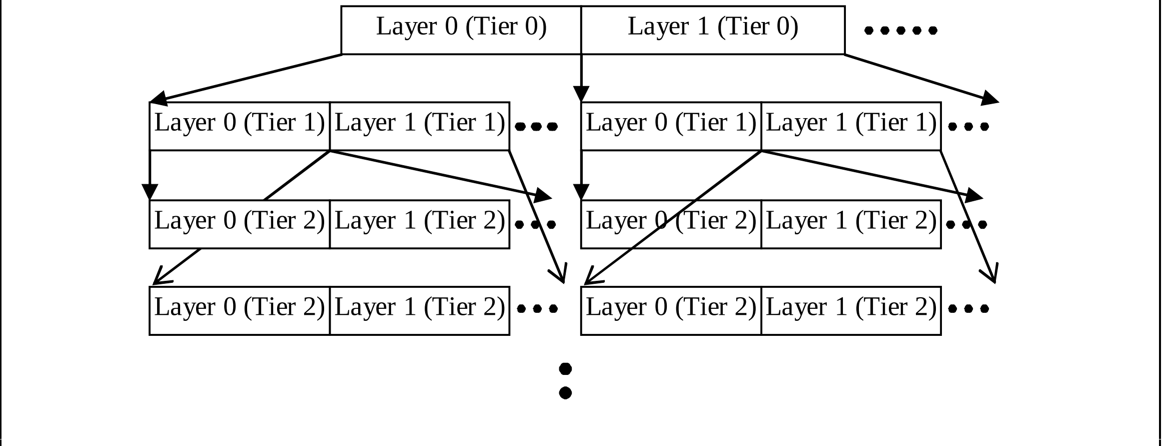


Figure 4. Meta-formats with nested scalability

alternative scalability progression mode – LRCP – the highest tier is SNR, and within SNR layers there are nested resolution layers, followed by color layers and precinct layers. However, the multi-tier nested scalability structure is common in both.

The above-described logical meta-format is analogous to that of a book, where there are nested layers for chapters, sections, sub-sections and so on. It is conceivable that the book-format be common across all books irrespective of content. Likewise, all scalable bit-stream representations can be cast into a common nested scalability structure that can be standardized into a bit-stream model, irrespective of content.

Note that the above nested structure is merely logical, in the sense that in the actual bit-stream there is more freedom in where the data segments lie. The layers at the deepest tier are called *atoms*. They consist of an arbitrary number of contiguous segments of the bit-stream that can be dropped as part of an adaptation process. There can also be arbitrary filler code in between the atoms, which are always included and never dropped as part of any adaptation.

3.2 Notation and Data cube representation

Formalizing the notation for the bit-stream, if the data has L nested tiers of scalability, and the *i*th tier contains l_i layers, we can say that the data consists of a concatenation of $l_0 \times l_1 \times \ldots \times l_{L-1}$ data chunks $B(j_0, j_1, \ldots, j_{L-1})$, where $j_0=0,1,\ldots, l_0-1$; $j_1=0,1,\ldots, l_1-1$; …; $j_i=0,1,\ldots, l_i-1$;…; $j_{L-1}=0,1,\ldots, l_{L-1}-1$. A way to visualize this data is to consider a L-dimensional *data cube* of size $l_0 \times l_1 \times \ldots \times l_{L-1}$, the $(j_0, j_1, \ldots, j_{L-1})^{th}$ element of which is the logical data chunk $B(j_0, j_1, \ldots, j_{L-1})$, called the *atom*. The full bit-stream is essentially a concatenation of these atoms in any order. Note that in the data cube representation, there is an assumption of symmetry in the nested scalability, but considering that an asymmetric structure can be converted into a symmetric one by inclusion of empty atoms, this is not a restriction by any means.

Using an example of the first two tiers of JPEG2000 RLCP progression mode, we can visualize the data as organized in a 2-dim cube (L = 2) as shown in Figure 5. The bulk of the bit-stream apart from any filler code can be visualized as being obtained by scanning the atoms in the data cube in some order. The same concept generalizes readily to more

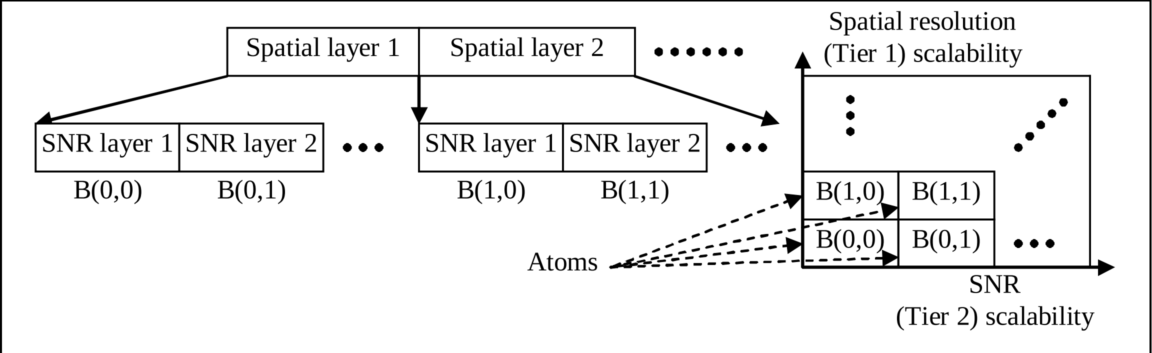


Figure 5. JPEG2000 example bit-stream

than two dimensions or nested tiers. An example of a three-dimensional data cube is shown in Figure 6.

3.3 Model based Adaptation

3.3.1 Models

The data cube representation defined above applies both to true scalable bit-streams where successive layers in each tier are handled incrementally by a decoder, or to the case when one or more tiers are handled exclusively. The latter is essentially equivalent to multi-version scalability, where multiple independent versions are maintained simultaneously in the layers of these tiers, but an eventual recipient would use only one of them. Generalizing, each tier in the meta-bit-stream format can be either *incremental* or *exclusive* in terms of scalability. The descriptor contains a flag for each tier to denote whether the tier is multi-version or incremental. If all tiers are *exclusive*, the bit-stream is fully multi-version with each atom being an independent version. If all tiers are *incremental*, the bit-stream is truly scalable (Eg. JPEG2000). In the most general case, tiers could be mixed between *incremental* and *exclusive* scalability.

Furthermore, in order to cater to certain situations related to multiple description

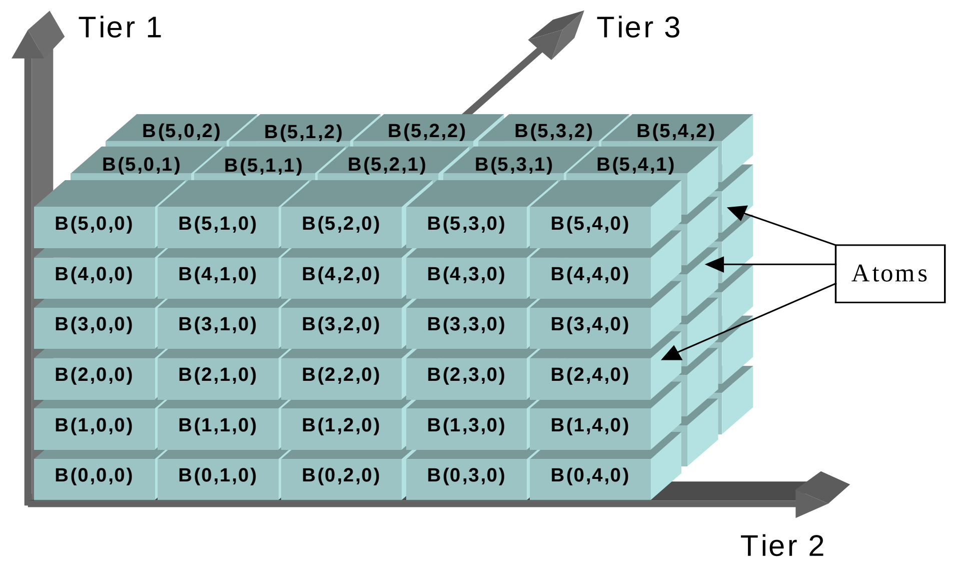

Tier 1
Tier 3
Tier 2
Atoms
B(5,0,2) B(5,1,2) B(5,2,2) B(5,3,2) B(5,4,2)
B(5,0,1) B(5,1,1) B(5,2,1) B(5,3,1) B(5,4,1)
B(5,0,0) B(5,1,0) B(5,2,0) B(5,3,0) B(5,4,0)
B(4,0,0) B(4,1,0) B(4,2,0) B(4,3,0) B(4,4,0)
B(3,0,0) B(3,1,0) B(3,2,0) B(3,3,0) B(3,4,0)
B(2,0,0) B(2,1,0) B(2,2,0) B(2,3,0) B(2,4,0)
B(1,0,0) B(1,1,0) B(1,2,0) B(1,3,0) B(1,4,0)
B(0,0,0) B(0,1,0) B(0,2,0) B(0,3,0) B(0,4,0)

Figure 6. Data cube representation of multi-tier scalable media

coding, we allow the descriptor to specify some variations of exclusive tiers. Specifically, an exclusive tier may be one of types: *single*, *firstAlways*, *lastAll*, *firstAlwayslastAll*. For an exlcusive tier of type *single*, any one layer in the tier is required for any adapted version. For exclusive tier type *firstAlways*, the first layer in the tier is always needed in additon to one other layer. For type *lastAll*, the last layer if included requires inclusion of all other lower layers, but all other adaptations behave as type single. Finally, for exlusive tier type *firstAlwayslastAll*, the *firstAlways* and *lastAll* behaviors are combined.

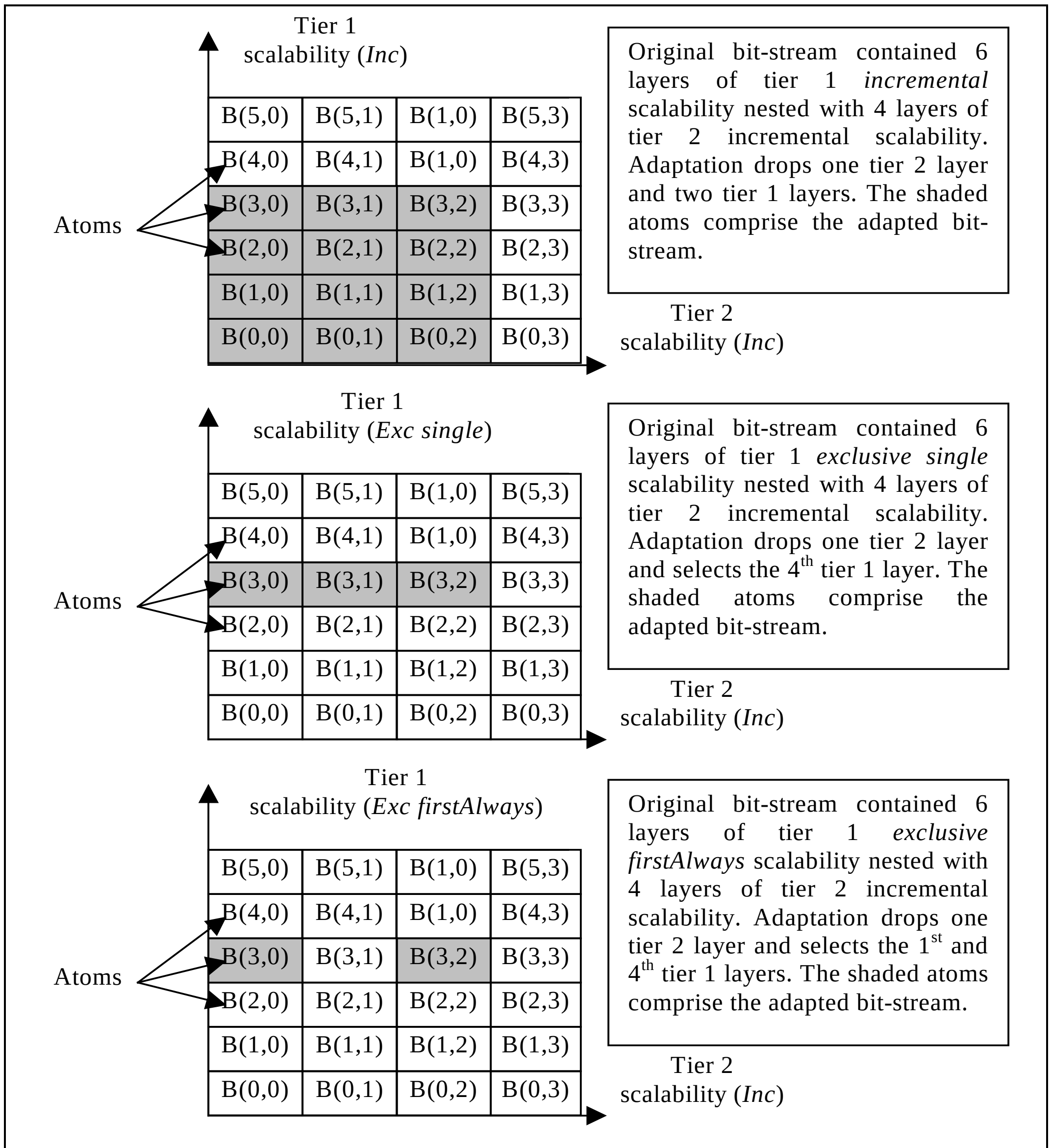
3.3.2 Adaptation

We next define formally adaptation operations on the multi-tier scalable bit-stream as defined above. The multi-tier data cube representation not only allows multiple dimensions of scalability to co-exist in a bit-stream, but also enables a simplified form of representation of adaptation based on the models mentioned above. Generally, with a scalable bit-stream conformant with SSM, any adaptation is simply implemented as dropping atoms, repacking the bit-stream, updating any TOCs appropriately, and doing other minor editing operations, while preserving the generic multi-tier structure so that it can be re-transcoded. For incremental tiers, layers can only be dropped from the outer end whereas for exclusive tiers, a specific subset of layers as given by the exclusive type, is dropped.

In SSM modeling, there are as many adaptation possibilities as there are coordinates in the data cube representation. Each set of coordinates maps to a specific adaptation. In addition, there is one adaptation possibility corresponding to null adaptation. Using our previous notation, if there are L tiers in an SSM component with the number of layers is tier i being l_i, then an *adaptation point* is denoted by the L-tuple (d_0, d_1,…, d_{L-1}), where $1 \leq d_i \leq l_i$. In addition, there is a null adaptation point (0, 0,…, 0), to make the total number of adaptation possibilities $1+ l_0 \times l_1 \times \ldots \times l_{L-1}$.

The ith component d_i of the adaptation point, correspond to tier i, $i = 0,1,\ldots, L-1$, and indicates to an adaptation engine one of the following: (1) If the ith tier is incremental, d_i layers from the beginning are included; (2) If the ith tier is exclusive single only the d_i^{th} layer is included in the ith tier; (3) If the ith tier is exclusive firstAlways, the first and the d_i^{th} layers are included in the ith tier; (4) If the ith tier is exclusive lastAll, then if the tier has exactly d_i layers, all of them are included but if it has more than d_i layers then only the d_i^{th} layer is included in the ith tier; (5) If the ith tier is exclusive firstAlwaysLastAll, then if the tier has exactly d_i layers all of them are included but if it has more than d_i layers then the first and the d_i^{th} layers are included in the ith tier. (6) If d_i=0 for all i (null adaptation point) then all layers are dropped from all tiers.

The adapted subset bit stream that reaches the decoder would then be given by some form of concatenation of the atoms B(j_0, j_1,…, j_{L-1}), where for each tier $i = 0,1,\ldots, L-1$, one of the following are included: (1) If the ith tier is incremental, $j_i = 0,1,\ldots, d_i-1$; (2) If the ith tier is exclusive single $j_i=d_i-1$; (3) If the ith tier is exclusive firstAlways, $j_i=0, d_i-1$; (4) If the ith tier is exclusive lastAll, $j_i = 0,1,\ldots, d_i-1$ if the tier has exactly d_i layers, and $j_i=d_i-1$ if the tier has more than d_i layers; (5) If the ith tier is exclusive firstAlwaysLastAll, $j_i = 0,1,\ldots, d_i-1$ if the tier has exactly d_i layers, and $j_i=0, d_i-1$ if the tier has more than d_i layers. (6) If d_i=0 for all i (null adaptation point) then all atoms are dropped in the adapted version.


Tier 1 scalability (*Inc*)
Atoms
B(5,0) B(5,1) B(1,0) B(5,3)
B(4,0) B(4,1) B(1,0) B(4,3)
B(3,0) B(3,1) B(3,2) B(3,3)
B(2,0) B(2,1) B(2,2) B(2,3)
B(1,0) B(1,1) B(1,2) B(1,3)
B(0,0) B(0,1) B(0,2) B(0,3)
Tier 2 scalability (*Inc*)
Tier 1 scalability (*Exc single*)
Tier 2 scalability (*Inc*)
Tier 1 scalability (*Exc firstAlways*)
Tier 2 scalability (*Inc*)

Original bit-stream contained 6 layers of tier 1 *incremental* scalability nested with 4 layers of tier 2 incremental scalability. Adaptation drops one tier 2 layer and two tier 1 layers. The shaded atoms comprise the adapted bit-stream.

Original bit-stream contained 6 layers of tier 1 *exclusive single* scalability nested with 4 layers of tier 2 incremental scalability. Adaptation drops one tier 2 layer and selects the 4th tier 1 layer. The shaded atoms comprise the adapted bit-stream.

Original bit-stream contained 6 layers of tier 1 *exclusive firstAlways* scalability nested with 4 layers of tier 2 incremental scalability. Adaptation drops one tier 2 layer and selects the 1st and 4th tier 1 layers. The shaded atoms comprise the adapted bit-stream.

Figure 7. Visualization of layer drops for 2-tier examples.

Note that if the transmitted data-stream has to be a non-null adaptation, in all tiers at least one layer must be transmitted. For null adaptation corresponding to adaptation point (0,0,…,0), none of the atoms are included.

Using the data cube visualization, dropping layers from the end in an incremental tier is equivalent to chopping off the ends of the data cube in units of layers. Selecting particular layers from an exclusive tier is equivalent to extracting slices from the data cube. In general, a reduced cube from the original is transmitted after adaptation. Some examples for the case of 2 nested tiers are shown in Figure 7.

3.3.3 Satellite Atoms

Regular atoms in the data cube representation are identified by coordinates, as in B(j_0, j_1, …, j_{L-1}), where $0 \le j_i \le l_i–1$. In addition, it is possible to allow one or more coordinates of an atom to be designated as don't cares, such as by making these coordinates –1. Such atoms are called *satellite* atoms. In any adaptation, if atom B(j_0, j_1, …, j_{L-1}) is to be included in the adapted version, all its satellite atoms, formed by replacing one or more coordinates j_i by –1, are also included bvy default. This explains the name satellite. Note however, that from the nature of the definition, each satellite atom is invariably a satellite of many regular atoms.

The purpose of satellite atoms is to allow convenient inclusion of common bit-stream segments in several possible adaptations, without needing to expand the number of layers in a tier, or the number of adaptation possibilities.

3.4 Causality Requirement

Because adaptation can be implemented as simple dropping of layers, an adaptation engine does not need to *decode* or *decrypt* content in order to perform adaptation. However, an encoder or an encrypter must maintain causality of the data atoms, so that a decoder or decrypter can still handle adapted content. In general, it is necessary to ensure that there are no dependencies across layers beyond that imposed by the adaptation model. Thus, the dependency across layers in incremental tiers is limited to being causal. For exclusive tiers, the dependency is still causal, but must be limited to those atoms that would be included in an adaptation, based on the specific exclusive type.

Specifically, the causality constraint for encoding ensures that for encoding data atom B(j_0, j_1, …, j_{L-1}), the encoder only uses information from atoms B(k_0, k_1, …, k_{L-1}), within the usual limits $0 \le j_i$, $k_i \le l_i - 1$ with at least one $k_i \ne j_i$, such that for incremental tiers i, $k_i \le j_i$, and for exclusive tiers i, $k_i=j_i$ and/or $k_i=0$ and/or $k_i \le j_i$ depending on the model and number of layers in the tiers. This ensures that for any usable adaptation based on the models, the decoder at the receiving end can decode the content unambiguously.

The causality constraint for encryption is that the starting state of the encryption engine for atom B(j_0, j_1, …, j_{L-1}), is derived from the ending states of the encrypter for adjacent causal atoms B(k_0, k_1, …, k_{L-1}), within the usual limits $0 \le j_i$, $k_i \le l_i - 1$ with at least one $k_i \ne j_i$, such that for incremental tiers i, $0 \le j_i - k_i \le 1$, and for exclusive tiers i, either $0 \le j_i - k_i \le 1$ or $k_i=0$, depending on the model and number of layers in the tiers. Progressive encryption enabling adaptation without decryption has been considered in [28], [29].

3.5 Parcels and components

The discussion so far essentially pertained to a single coded scalable *component*. A component is a coded unit of data that may be represented in a scalable logical bit-stream-format represented by a data cube. However, in a composite bit-stream, multiple coded components can co-exist. For example, one component may be an image and a second component may be audio annotation that goes with it; both components may be packaged together in the bit-stream to provide an experience of image viewing with audio annotation. Such a combination of components is called a *parcel*.

Generally, a *parcel* is a super construct of components that essentially define adaptation boundaries. It may be comprised by multiple *independent* scalable components to provide a composite experience. The overall bit-stream consists of multiple parcels, often all of the same type. Parcels are adapted almost independently and often sequentially in an adaptation engine, with limited dependency between successive parcel adaptations. The size of a parcel is really a design choice, and may range from an entire scalable compressed file to a network transmission packet. Continuing with our previous example of image with audio annotation, both the image and audio components may constitute a parcel, but there may be multiple parcels in a composite bit-stream to produce a slide show with audio. An alternative example arises in scalable video coding where each Group of Pictures (GOP) is represented independently as a scalable component. If GOPs are to be adapted independently, then the parcel is the GOP, and contains a single component.

It is typical for different parcels in the same bit-stream to be comprised by the same components. However, the encoding structure for each component may vary from parcel to parcel, depending on characteristics for the specific content.

Finally we note although the parcel construct in SSM denotes a unit of adaptation, its significance goes beyond that. The entire adaptation framework built around the parcel is designed to fit a variety of delivery models with little or no adjustments. For example, SSM accommodates the typical streaming scenario where information for each parcel from both the descriptor side as well as the recipient side, along with the parcel bit-stream, comes into an adaptation engine sequentially. Alternatively in an interactive application, parcels may be adapted and delivered randomly based on user interaction.

3.6 Bit-stream layout

Parcels, components and atoms within a component are essentially logical constructs that may exist anywhere in the bit-stream. While to make SSM adaptation viable for a given bit-stream, these constructs must exist and be defined, it is not necessary to impose syntactic restrictions on the bit-stream based on this hierarchy. An example of a bit-stream segment with two parcels, each consisting of two components is shown in Figure 8.

4. Adaptation variables

4.1 General

Having have seen how a generic scalable bit-stream looks like, and what an adaptation operation on it involves, we next need to talk about ways an adaptation engine can decide on an appropriate adaptation point, without knowledge of the specifics of the media and the coding. An adaptation engine is expected to have knowledge of certain relevant scalability properties of a SSM resource through the *resource description* XML. At the same time it expected to know the capabilities and preferences of its outbound connection through the *outbound constraints* XML specifications. The bridge between the two on the media creator/originator side and the receiver side of the adaptation engine is established through *adaptation variables*. The *resource description* and the *outbound constraints* speak the same language through these variables, so that an adaptation engine can decide how to drop layers to match the two sides.

If a receiver knew exactly the structure of the content it expects to receive through an adaptation engine, it could exactly specify the requested adaptation point in the engine's *outbound constraints* specifications. However, to assume that to be always the case is too restrictive. For example, if a receiver is expecting JPEG2000 images, but does not know what the dimensions and encoding parameters are for a particular image, it is not possible for it to request a specific adaptation point based on considerations of display resolution,

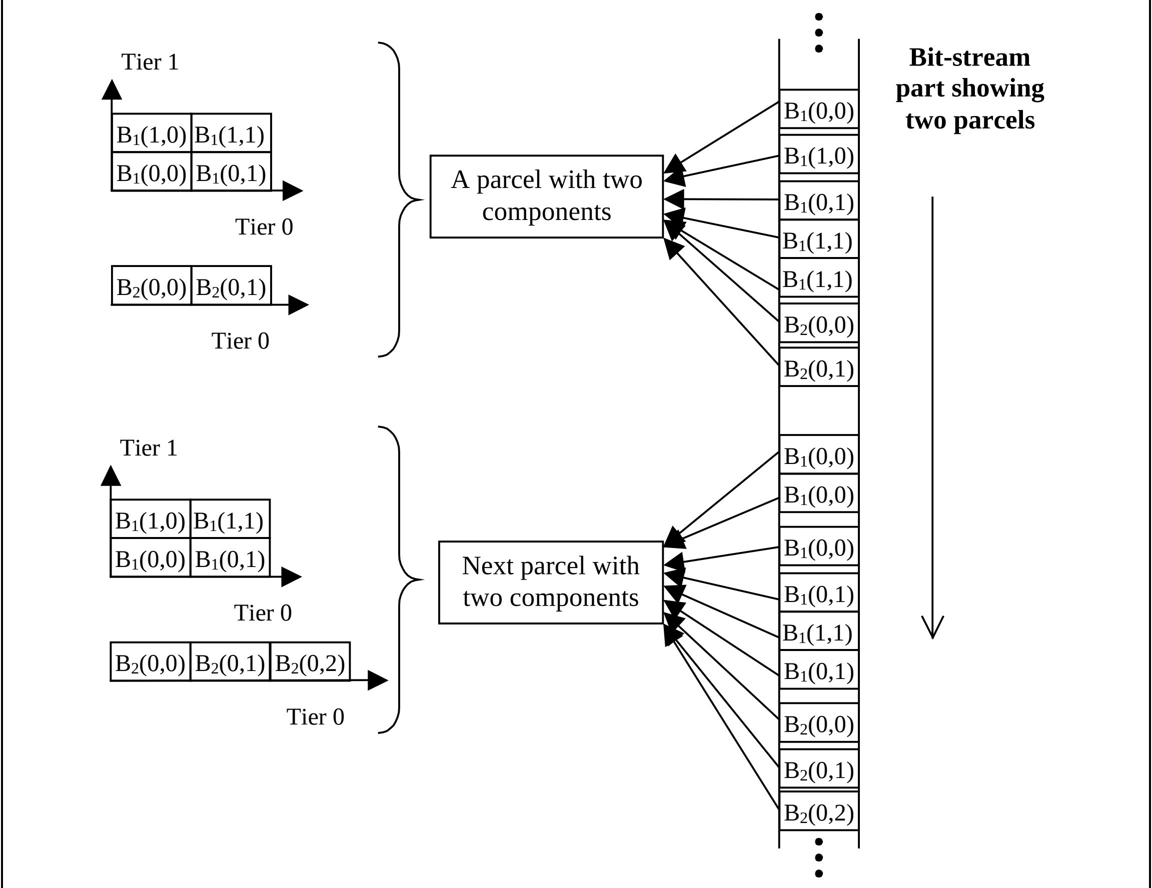


Figure 8. Bit-stream layout example for two parcels each containing two components

quality, and so on. That is why, it is important to dissociate the *resource description* and the *outbound constraints* from the structure as much as possible, and *adaptation variables* allow us to achieve that.

The most important property of all adaptation variables is that they are expressed quantitatively in terms of *non-negative* (floating) numbers, referred to as *variable values,* defined over the discrete space of all possible adaptation choices. Whatever method is used to quantify the variables must be communicated to the developer of the media experiencing system. However, an adaptation engine itself does not need to know what they mean. Values have different interpretations for the media creator, the consumer, and the adaptation engines in between. To the media creator/originator, they are quantified properties based on which a content may be adapted. To a media consumer they are quantified properties to indicate its limitations and preferences. To an adaptation engine, they are simply numbers based on which it must decide how to drop layers and adapt an input bit-stream.

4.2 Feature variables

4.2.1 Definition

Feature variables are certain quantifiable properties relevant to the experience of a single media component or jointly for a set of media components. Features defined for a single component are called *elemental features*, while those defined over more than one component are called *product features*. Some examples of elemental feature variables are: *Codesize, MeanSquaredError, SpatialResolution, TemporalResolution etc.* One example of a product feature variable is: *PerceptualRichness* which is a product feature of the adaptation points of audio and image components of a parcel, but which cannot be expressed as a function of individual features from the two components.

4.2.2 Identification

Each feature is associated with a name that uniquely identifies the feature. The uniqueness is within the context of the media parcel being adapted/delivered. Thus, the feature names used in the resource description XML and the outbound constraints XML for the same parcel of media must be consistent, but across different media-types or parcels there is no restriction on the names used because when the names are resolved at the adaptation engine there is no scope of a conflict.

The media creator, who provides the resource description XML, defines features relevant to the media the way he/she chooses, and communicates their unique names, meanings, and value spaces to the media experiencing system developer so that the latter can generate meaningful outbound constraints.

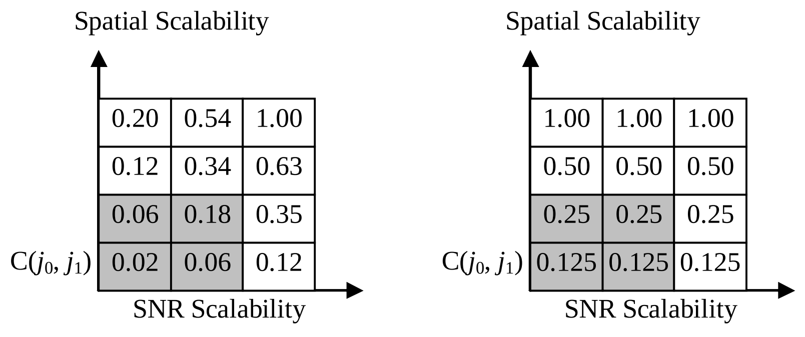
4.2.3 Values and Distribution

The resource description XML conveys for each feature the quantitative values the feature would have for all possible *adaptation points* of the SSM components over which it is defined. This set of values is referred to as the *feature distribution*. We first consider elemental features, and next consider product features.

4.2.3.1 Elemental Features

If there are L nested tiers in a component with l_i layers in the ith tier, it is necessary to provide a L-dimensional matrix of size $l_0 \times l_1 \times \ldots \times l_{L-1}$, whose $(j_0, j_1, \ldots, j_{L-1})^{\text{th}}$ element denoted $C(j_0, j_1, \ldots, j_{L-1})$, for $j_0 = 0,1,\ldots, l_0-1$; $j_1 = 0,1,\ldots, l_1-1$; …; $j_i = 0,1,\ldots, l_i-1$;…; $j_{L-1} = 0,1,\ldots, l_{L-1}-1$, is a non-negative number specifying the value of the feature if $(j_0, j_1, \ldots, j_{L-1})$ is the adaptation point, along with an *empty* feature value C_φ specifying the feature value the component would have when the entire component is dropped, *i.e.* none of the layers are transmitted. The total number of values that need to be sent is therefore $1 + l_0 \times l_1 \times \ldots \times l_{L-1}$. In practice, all these values are specified with respect to a reference feature value for convenience. In this case, the elements $C(j_0, j_1, \ldots, j_{L-1})$ multiplied by the reference value provides the true feature value for adaptation point $(j_0, j_1, \ldots, j_{L-1})$. In any case, the reference multiplied by the last fraction $C(l_0-1, l_1-1, \ldots, l_{L-1}-1)$ yields the *full feature value*, or the value the feature would have if the content were transmitted without any layer-dropping adaptation for incremental tiers and with the highest layer versions included for exclusive tiers. The same principle of multiplying with the reference applies to the empty feature value C_φ.

From the nature of features that may typically be defined, it is often the case that the distribution is either monotonic non-increasing or monotonic non-decreasing. For example, *Codesize* (rate) is a feature that is always monotonic non-decreasing. For a monotonic non-decreasing type feature, the values $C(j_0, j_1, \ldots, j_{L-1})$ would be analogous to the cumulative distribution of a multi-dimensional discrete random vector, if the



(a) Feature: *Codesize* (b) Feature: *DisplayResolution*

Figure 9. Exact Distribution Specification for JPEG2000 example

reference multiplier value is the same as the full feature value. This explains the use of the term distribution.

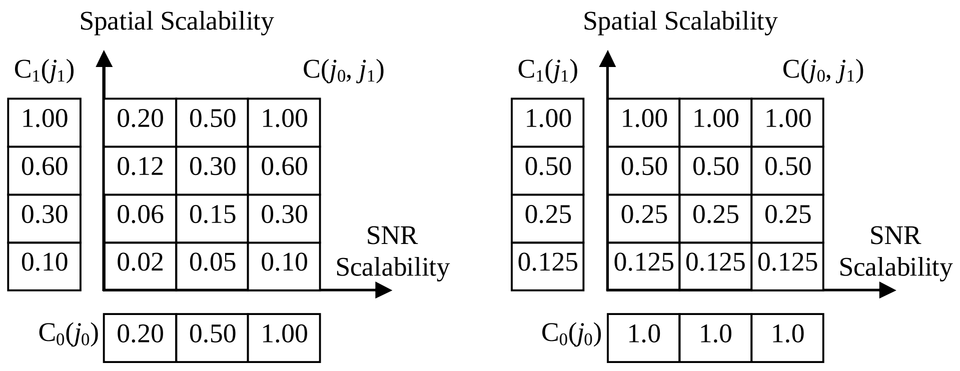
For example, considering the first two tiers of JPEG2000 RLCP progression mode, the distribution specifications for features *Codesize* and *DisplayResolution* may look as in Figure 9. Both are non-decreasing monotonic. Here we have four spatial scalability layers nested with three SNR scalable layers each. Note that in Figure 9(b), the *DisplayResolution* attribute does not change with SNR scalable layers. As a result of transcoding, if a SNR layer and two spatial layers are dropped, the *Codesize* attribute of the transcoded bit-stream shown shaded in Figure 9 would be 0.18 times the reference *Codesize* value, while the *DisplayResolution* attribute would be 0.25 times the reference *DisplayResolution* value.

Oftentimes, it is be more convenient and less expensive in terms of overheads to express the cumulative distributions only approximately using products of one or more individual lower-dimensional *marginal* distributions. In this case, the element $C(j_0, j_1, \ldots, j_{L-1})$ is obtained approximately as $\hat{C}(j_0, j_1, \ldots, j_{L-1})$ using a product combination of marginal distributions. That is, the specification involves P lower dimensional cumulative distributions $C_i(.)$ that cover L dimensions together: $\hat{C}(j_0, j_1, \ldots, j_{L-1}) = C_0(\ )\times C_1(\ )\times\ldots\times C_{P-1}(\ )$. The empty feature C_φ is transmitted separately.

For the JPEG2000 example of Figure 10 the approximate specifications using two one-dimensional marginals and the eventual approximate distributions generated are shown in Figure 10. As seen in Figure 10 (b), the *DisplayResolution* feature has been represented exactly using the approximate approach, while the *Codesize* feature in Figure 10(a) is represented only approximately.

4.2.3.2 Product features

The resource description XML conveys for each product feature the quantitative values the product feature would have for all possible joint *adaptation points* of the SSM components involved in the product feature. This set of non-negative values is the product *feature distribution*. If there are C components involved in the product feature, with L^c nested tiers in the *c*th component ($c = 0,1,\ldots,C-1$) it is necessary to provide 2^C-1 non-empty distributions corresponding to the case where at least one component is non-empty (included), along with an all empty feature value C_φ corresponding to the case



$C_1(j_1)$	$C(j_0, j_1)$		
1.00	0.20	0.50	1.00
0.60	0.12	0.30	0.60
0.30	0.06	0.15	0.30
0.10	0.02	0.05	0.10
$C_0(j_0)$	0.20	0.50	1.00

(a) Feature: *Codesize*

$C_1(j_1)$	$C(j_0, j_1)$		
1.00	1.00	1.00	1.00
0.50	0.50	0.50	0.50
0.25	0.25	0.25	0.25
0.125	0.125	0.125	0.125
$C_0(j_0)$	1.0	1.0	1.0

(b) Feature: *DisplayResolution*

Figure 10. Approximate Distribution Specification for JPEG2000 example

when all components are empty. Among the non-empty distributions there is one corresponding to the case where all components are included. In this case, the distribution specifies a ($L^0+L^1+\ldots+L^{C-1}$)-dimensional matrix with number of elements equal to the product $\prod (l^c_i)$ over $c = 0,1,\ldots,C–1$ and $i = 0, 1, \ldots, L^c–1$, where l^c_i is the number of layers in tier i of component c. There are other 2^C-2 non-empty distributions corresponding to the cases when one or more components but not all are empty. Any such partial empty distribution is specified as a reduced dimensional distribution over the non-empty components. The total number of values that need to be sent comprising all the non-empty distributions and the empty value is the product $\prod(1 + l^c_0 \times l^c_1 \times \ldots \times l^c_{L^c-1})$ over all c.

Each non-empty distribution can be individually specified using a product of marginals as in the elemental feature case.

4.3 Component variables

4.3.1 Definition

Just as features have unique names so do components within the context of a parcel. The media creator not only conveys feature variables, their meanings and value spaces to the experiencing system developer but also the names of components. Based on the component name, certain variables are defined by default. These are called component variables.

4.3.2 Indicators and Values

The first variable family defined by default is the *inclusion indicator*. This variable has a value of 1 for all non-empty adaptation points, and zero only if all the atoms are dropped. A component is assumed to be included if at least one of its atoms is included. This variable can be used to specify complex constraints based on inclusion or exclusion of whole components, for e.g. if a certain component is included another one must be included, and so on.

The second family of indicators is called *layers in tier indicators*, which convey the number of layers in the adaptation point for a specified tier index parameter. Thus if the adaptation point is (j_0, j_1, …, j_{L-1}), then the value of this variable corresponding to tier i is j_i.

The third family of indicators is called the *current number of layers in tier indicators*, which conveys a constant whose value is the total number of layers currently in the bit-stream for a specified tier index parameter.

The fourth family called the *original number of layers in tier indicators,* which conveys a constant whose value is the original number of layers in the bit-stream for a specified tier index parameter, before any adaptation step.

4.4 Combination variables

4.4.1 Definition

Sometimes the media creator can define combination variables in the resource description XML, which are essentially mathematical real and/or Boolean expressions and functions involving feature variables, component variables or other combination variables from a variety of components. The combination variables are conveyed to the experiencing system developer in the same way as feature variables, and serve as a short cut for the outbound constraints XML.

Each combination variable may be associated with a certain number of arguments during specification so that the variable can be used as a function rather than as a static expression.

One example of combination variables is *TotalCodesize,* which may be defined as the sum of the *Codesize* features for individual components in a parcel. Another example, involving the component inclusion indicator variables is a Boolean expression that indicates if component_1 is included, component_2 must be included (x=> y is equivalent to x′+ y). A third example, takes one argument, and computes a *polynomial* at the value given by the argument.

4.4.2 Identification

Combination variable are identified in the same way as feature variables, i.e. with a unique name. The resource description XML provides a name for each combination variable as they are defined.

4.4.3 Expression specification and evaluation

The mathematical expression for each combination variable is specified in the resource description XML by means of an ordered list of numeric constants, adaptation variables, arguments and operators that must be pushed into an expression stack for evaluation of the expression. Variables pushed into the stack can be feature variables, component variables or previously defined combination variables, each identified by its unique name. Operators pushed into the stack can take any number of operands. When a combination variable takes arguments, the definition of the combination incorporates a means to reference the arguments of the combination function in order. This is the only situation where arguments can be pushed, i.e. during specification of a combination variable that takes arguments.

Evaluation of an expression at an adaptation engine for a given set of adaptation points corresponding to components of a parcel is done as follows. When a constant is pushed its numeric value is pushed into the stack as a real numeric element. When a variable is pushed, the numeric value of the variable for the given set of adaptation points is evaluated, and pushed into the stack as a numeric element. When a unary operator is pushed into the stack, the current top operator element as well as the next top stack element, which must be a numeric one, are popped out immediately. The operator operates on the numeric operand, and the result is pushed back into the stack as a numeric element. When a binary operand is pushed into the stack, the current top operator element and the two next top stack elements, both of which must be numeric, are popped out immediately. The binary operator operates on the numeric operands, and the result is pushed back into the stack as a numeric element. The same methodology is used for n-ary operators.

When a combination variable taking arguments is called, a certain number of elements equal to the number of arguments taken by the function, are popped from the stack in order, and the combination is evaluated based on the definition.

When all the elements in the expression ordered list has been processed, the topmost stack element yields the value of the expression.

A small set of useful real and Boolean operators is allowed for forming expressions. When real and Boolean operators and operands are mixed, the following conventions are used to make the necessary transformations between the two domains: a Boolean 0 has real value 0.0, a Boolean 1 has real value 1.0, any real non-zero value has Boolean value 1, and a real zero 0.0 has Boolean value 0.

4.5 Look up table variables

While in most cases it is sufficient to work with feature variables that are tied to the structure of the content, in some cases, it may be necessary to map arbitrary recipient inputs into appropriate quantities related to structure, so that an optimum decision can be made. In many sitiations it may be reasonable to assume that the mapping from an arbitrary input space to a quantity related to the structure is already known by the receiving terminal. However, in some cases, the content creator may want to control this mapping. Look up table variables provide a tool in the descriptor XML to allow specification of an arbitrary scalar function $f(x_0, x_1,\ldots)$. Note that the same purpose can be served by combination variables used as functions (see Section 4.4), if is possible to find a good enough mathematical model for the function.

A look up table presents a set of values of a function, at points on a possibly non-uniform grid in a multi-dimensional input space. The values of the function at points not on the grid are interpolated using multi-dimensional linear interpolation. Consider a N-dimensional function $f(x_0, x_1,\ldots,x_{N-1})$, where the grid on the ith dimension has $m(i)$ points in ascending order: $\{g_{i,0}, g_{i,1},\ldots, g_{i,m(i)-1}\}$. Then the look-up-table is specified using a set of $\prod (m(i)+2)$ values over all i, obtained by extending the grid on each dimension in lower and upper directions yielding two virtual grid points. Grid extension allows convenient specification of values of the underlying function at values outside the support region given by the grid. Any input that is outside the support region in any dimension given by the real grid points, snap to the virtual grid on the low or high side, before interpolation is invoked. Note that in this technique it does not really matter where the virtual grid points are actually located, of a look-up-table in two dimensions is shown in Figure 11.

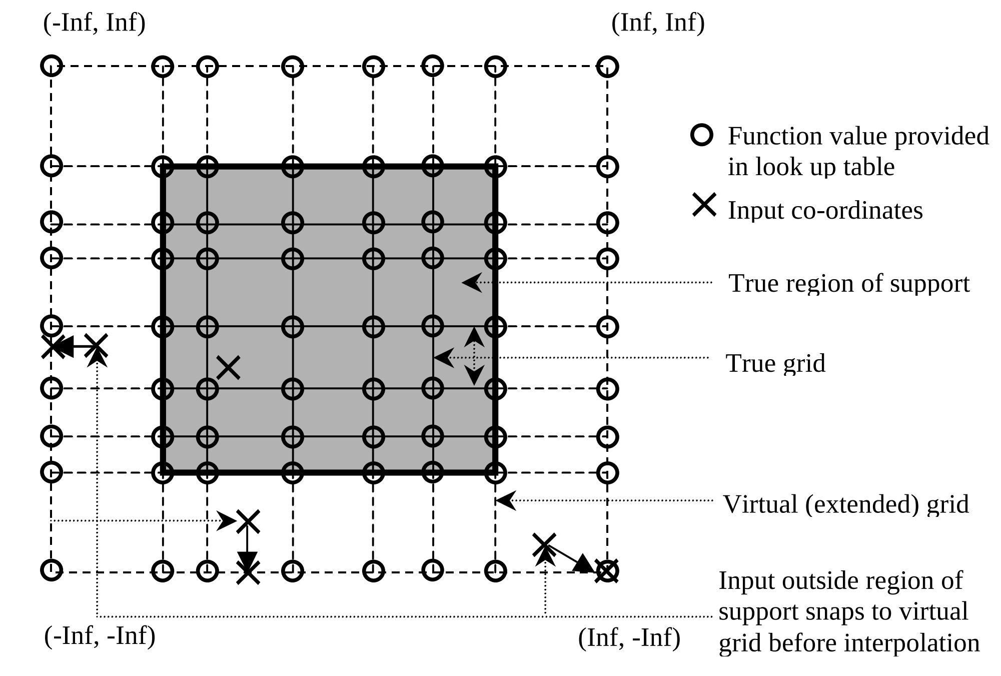


Figure 11. Look-up-table example in two dimensions

When a look-up-table variable is called, it takes as many arguments as there are dimensions in the table, with the arguments corresponding to the co-ordinates of the point at which the underlying function is to be evaluated. The arguments are popped from the stack in order, and the look-up-table is evaluated by either multi-dimensional linear interpolation, or by one of the methods: *rounding to nearest grid point*, *ceiling of grid cell*, and *floor of grid cell*, where each co-ordinate is respectively rounded, ceiling-ed or floor-ed with respect to the grid points.

4.6 Reserved adaptation variables

In the discussion so far, we have been talking about so called *custom* or content specific adaptation variables that can be defined and named at will by content creators so that the experiencing system at the receiving end can issue appropriate outbound constraints based on them. It is worth mentioning however that there is value in standardizing the identifying names and the quantification method for certain variables with universal meaning across different types of content. These variables are called *reserved adaptation variables*, because their identifying names are effectively reserved and disallowed for use as custom adaptation variables. Examples of adaptation variables that could be standardized in name and quantification method are: *Bandwidth*, *NumerOfSpeakers*, *DisplayResolution*, *ProcessorSpeed* etc. Use of reserved variables enable creation of *content independent profiles* for terminals, that allow adaptation say at an edge server, without an explicit request from a receiving terminal in the form of outbound constraints, for every type of content that is received. In other words, it enables passive reception of various types of content by uploading a stationary content-independent profile involving reserved variables to an adaptation engine once and for all.

Reserved adaptation variables may be either feature variables or combination variables defined differently for different types of content. But the eventual meaning and the value space of the adaptation should be the same for different content types.

Note that since the SSM framework per se does not change depending on whether an adaptation variable is custom or reserved, we do not dwell on this issue further.

5. The XML contents

5.1 Resource Description XML

Because the resource description XML has mostly been covered already in the discussion so far, here we summarize the key points, and mention the additional features. This metadata originates from the media creator and contains a full description of the bit-stream that enables an adaptation engine to decide how to drop layers.

5.1.1 Scalability structure

The metadata specifies the complete hierarchical model of the bit-stream with parcels, components, and atoms, and where the atoms lie in the bit-stream. For each parcel it defines a set of elemental and product feature variables and specifies their distributions, as well as a set of combination variables that apply locally within the parcel. It also defines global combination variables that apply to all parcels.

5.1.2 Limit constraints from content creator

Besides, defining adaptation variables, the metadata also includes constraints that are to be enforced by an adaptation engine. These are directives from the content creator to restrain the adaptation choices, and apply either locally to a parcel or globally to all parcels. The type of constraints specified here is referred to as *limit constraints*, and would be elaborated in the next section. The adaptation engine combines the constraints specified by the content creator in the resource description metadata with those specified by the receiver in the outbound constraints specifications to obtain the full set of constraints that need to be satisfied by an adaptation point.

5.1.3 Resource edit information

The resource description metadata contains information pertaining to editing of the resource bit-stream based on adaptation decisions made for each parcel. For example, it may often be necessary to modify information such as number of layers included and so on in the bit-stream, after adaptation. The metadata specifies for each parcel exactly where in the bit-stream a certain number of bits have to be replaced after the decisions have been made and the adaptation has been conducted, how many bits the replaced value spans and what its endian order is, as well as what the modified value is. The modified value is given by a stack expression as in Section 4.4.3. The output length in bits can be specified through a constant or through a feature variable. Note that this protocol actually allows a wide range of bit-stream modifications based on adaptation decisions. Even when it is not possible to have expressions to evaluate the modified value, the content creator can always define feature variables, one for the content and another for the length to denote what the correct bit-stream should be for each adaptation possibility.

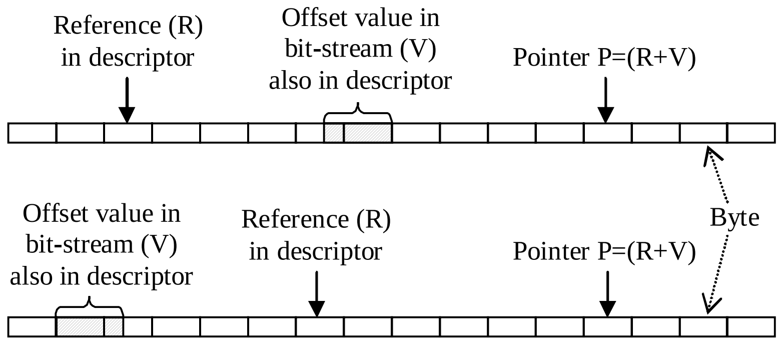

Reference (R) in descriptor
Offset value in bit-stream (V) also in descriptor
Pointer P=(R+V)
Byte
Offset value in bit-stream (V) also in descriptor
Reference (R) in descriptor
Pointer P=(R+V)

Figure 12. Codec offset data

5.1.4 Codec offset information

Sometimes in a compressed bit-stream there may be pointers that specify locations of other parts of the bit-stream or lengths of certain bit-stream segments. When SSM atoms are dropped as part of the adaptation process it is likely that this location/length information will become invalid. Therefore, in order to keep the adapted bit-stream consistent and decodable it is necessary to modify the relevant fields in the bit-stream as and when atoms are dropped. Because it is the adaptation engine that must modify the bit-stream, it is necessary to provide this offset/location information in the resource description metadata. The resource descriptor allows specifying locations in the bit-stream where offsets occur.

Both locations and lengths can be conveniently expressed as offsets from a given reference point. As shown in Figure 12, the resource descriptor specifies a reference point in the bit-stream (R), the exact location in bit-stream, the length in bits and endian type of where the value of an offset field is stored in the bit-stream, along with the actual numeric value (V) stored in this field. The numeric value is redundant, but has been included in the descriptor for convenience of implementation. The values R and V together provide the location of another point P in the bit-stream, where P=R+V. Alternatively, the value V provides the length of a bit-stream segment from R through P including R but excluding P, or excluding R but including P.

Based on the above description conveying reference R, the location of an offset/length field in the bit-stream, and the value V stored therein, an adaptation engine can modify the field as and when bit-stream segments are dropped to update the value of the difference P-R. In this regard, certain situations may arise, which are worth noting since they should be explicitly handled in the right way for a given bit-stream. First, if the field where an offset is stored or a part thereof (since it can span multiple bytes) is dropped as part of adaptation, the entire entry corresponding to the field can be removed from the descriptor, because it does not need to be handled anymore. Next, consider the situation when either the byte at location R or the bytes at location P or both are removed as part of adaptation, but the field where the offset is stored still remains valid. For these situations, the descriptor should explicitly mention how the pointers are to be updated before the new value of V the offset could be computed and updated in the resource. The invalid pointers R or P could be either moved up to the next valid byte, or moved down to the previous valid byte, and the result of the updated value V would be different based on which one is done. There is also a third option that zeroes out the value V stored in the field when either R or P becomes invalid. The semantics of the offset field in a given bit-stream determines the most appropriate way to handle invalid pointers, but the descriptor should clearly mention the handling technique to be used.

As an example, consider a length field in the bitstream. If the offset field indicates the length of R through P including R but excluding P with R<P, then both R and P should be moved up when invalid. Alternatively, if the offset field indicates the length of R through P excluding R but including P with R<P, then both R and P should be moved down when invalid.

5.1.5 Sequence data information

Sometimes in the compressed bit-stream, there may be fields representing monotonic sequences. For example, there may be bit-stream fields containing sequential counters, such as packet number fields in data packets or frame number (temporal reference) fields associated with compressed video frames. When bit-stream segments are dropped as part of a format agnostic adaptation process, these counters need to be updated accordingly for consistency. The sequence data descriptor element as part of the resource description metadata, allows a compact specification of where the counter fields are and how they should be updated so that a format agnostic processing engine can process the metadata and update the counter fields correctly. While such update operations could be accomplished also by using resource edit information as described in 5.1.3, in order to improve the resource description compactness, operation efficiency, and ease of specification of this descriptor it is appropriate to introduce a specific tool for sequence data information in the resource description metadata.

The descriptor allows defining an arbitrary number of sequences. Each sequence has an arbitrary number of elements included within it. A sequence is also associated with a sequence value, that is incremented/decremented for each countable element included. The starting value and the step value for increment/decrement is specified for each sequence in the resource description metadata. Optionally a modulo value could also be specified. If the sequence value equals to or exceeds the modulo value, the remainder (modulus) obtained by dividing the sequence value by the modulo value, is used as the new sequence value. For example, if the starting value of the sequence is s, and the step value of the sequence is p, the modulo value is m, and there are n countable elements in the sequence, then after i countable elements ($i = 0,\ldots, n$), the sequence value is $(s+i\times p) \bmod m$.

The elements within a sequence, can be *countFields*, sub-*sequences, countOnly* elements, or *writeFields,* as described below.

CountFields are associated with a location in the bit-stream and length in bits, both of which are specified in the descriptor. A sequence can have sub-sequences embedded recursively within. The starting value of a sub-sequence could be absolute or relatively derived from the sequence value of the parent sequence. For example, if a sub-sequence has its starting value *start_val*, and the sub-sequence gets sequence value *seq_val* from the parent sequence, the actual starting value for the sub-sequence will be *seq_val+start_val* for relative start, and just *start_val* for absolute start. There can also be countOnly elements, that are dummy countable elements that are not associated with any bit-stream location. All countFields, conuntOnly elements, and non-empty sub-sequences in the sequence effect an increment/decrement in the sequence by the step value, starting from the start value of the sequence. In addition to countable elements, there can be writeFields that can occur at any position in the sequence, and writes either the current sequence value or the number of elements till the current position in the sequence. The writeField, however, is not counted, that is, the sequence value is not updated at the end of a writeField.

After an adaptation involving bit-stream segment drops, the countFields in the sequence may point to dropped data. When a countField points to a valid bit-stream location after adaptation, the current sequence value is written at the location associated with the countField, and the sequence value is updated at the end of the countField. Optionally, it is it is possible to only update the sequence value but not update the bit-stream field associated with it countField element. When the bit-stream location (or part thereof) associated with a countField is dropped during adaptation, the handling depends on whether the sequence has been designated *packable* or *non-packable*. Genrally speaking, in a packable sequence individual elements can be removed independently, but in a non-packable sequence only the sequence as a whole can be removed from the parent sequence. Thus, for an invalid countField after adaptation, if the sequence is packable, the countField is simply removed from the sequence; but if the sequence is non-packable, the countField is replaced with a countOnly element. That means, the sequence value will still be incremented for the dropped field. Only when all the elements in a non-packable sequence become countOnly, the sequence can be disposed off. It can be removed if the parent sequence is marked as packable, and replaced with a countOnly field if the parent sequence is non-packable. That means, the sequence value will still be incremented.

The above metadata allows a very flexible means for specification of various kinds of sequences that may be encountered in a scalable bit-stream, allowing convenient updates once atoms have been dropped by adaptation.

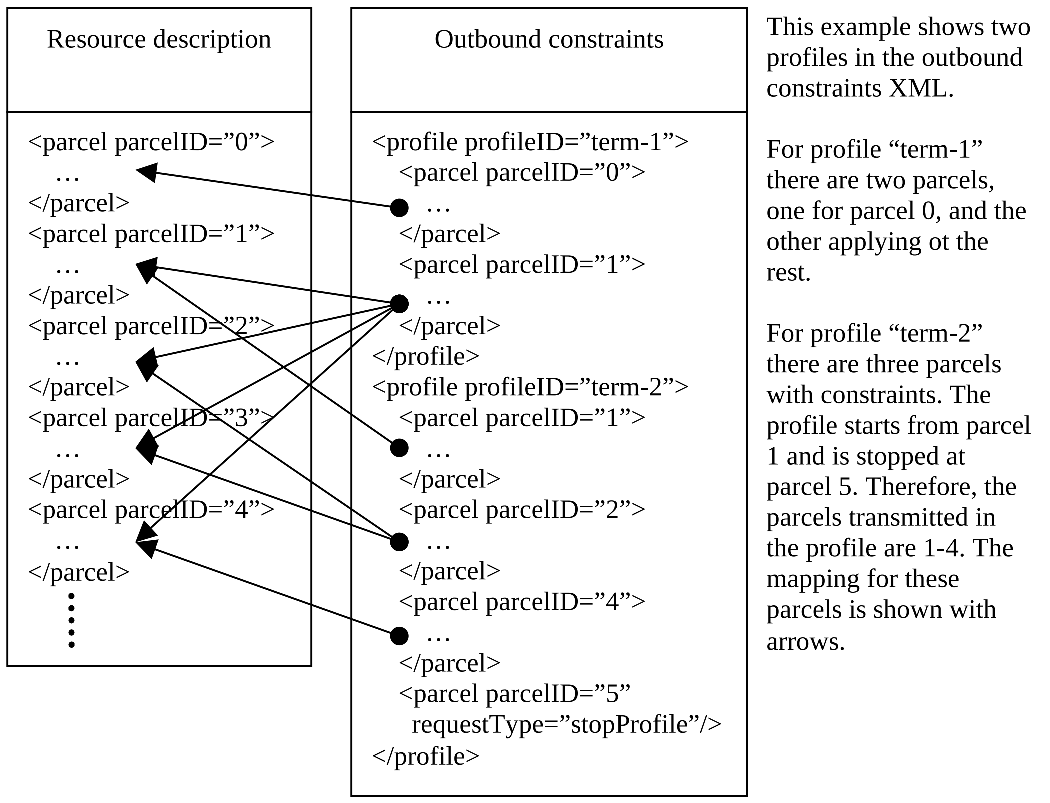
5.2 Outbound constraints XML and decision making

5.2.1 Parcel mapping for multiple recipient profiles

The outbound constraints XML provides a specific request to an adaptation engine to adapt the content before delivery. In general, the outbound constraints XML can have multiple profiles, one for each downstream recipient, each associated with an optional profile name. For each recipient profile, a set of constraints can be specified corresponding to each parcel.

In order to make decisions and adapt a bit-stream, an adaptation engine needs to have an outbound constraints specification for every parcel. Recall that a parcel is a unit of adaptation. In many situations however, it is unnecessary to provide a new set of constraints for every parcel. Constraints tend to remain the same for a large number of parcels, and only needs intermittent updates. In other words, the number of parcel constraints specified can be less than the number of parcels in the content bit-stream. The constraints are updated intermittently, and resource parcels are always adapted using the most current constraint specification.

Specifically, each parcel in the descriptor is associated with a parcelID that increases sequentially. Likewise, in the outbound constraints for each profile, there is a parcelID associated with each parcel, to indicate the resource and descriptor parcel to which the constraints apply. The parcelIDs for successive parcels in the outbound constraints for every profile must be in ascending order, but there can be arbitrary skips in between. The skipped parcels are assumed to have the same constraints as the preceding parcel.



```
<parcel parcelID=”0”>
  …
</parcel>
<parcel parcelID=”1”>
  …
</parcel>
<parcel parcelID=”2”>
  …
</parcel>
<parcel parcelID=”3”>
  …
</parcel>
<parcel parcelID=”4”>
  …
</parcel>
```

```
<profile profileID=”term-1”>
  <parcel parcelID=”0”>
    …
  </parcel>
  <parcel parcelID=”1”>
    …
  </parcel>
</profile>
<profile profileID=”term-2”>
  <parcel parcelID=”1”>
    …
  </parcel>
  <parcel parcelID=”2”>
    …
  </parcel>
  <parcel parcelID=”4”>
    …
  </parcel>
  <parcel parcelID=”5”
    requestType=”stopProfile”/>
</profile>
```

This example shows two profiles in the outbound constraints XML.

For profile “term-1” there are two parcels, one for parcel 0, and the other applying ot the rest.

For profile “term-2” there are three parcels with constraints. The profile starts from parcel 1 and is stopped at parcel 5. Therefore, the parcels transmitted in the profile are 1-4. The mapping for these parcels is shown with arrows.

Figure 13. Parcel mapping between resource description and outbound constraints

Furthermore, there is a provision to stop a profile at a given parceled by using a special termination code. A profile is assumed to commence at the parcelID corresponding to the first parcel specified within the outbound constraints XML for the profile, and terminate at the parcel with parcelID just less than the one that has a terminate profile code. Note that this framework allows recipients to sort of ‘tune in’ or ‘tune out’ of an ongoing broadcast at will.

This principle is shown by means of an example in Figure 13. As shown in the figure, the outbound constraints XML has multiple receiver profiles defined. The parcel-mapping rule applies individually to each profile. We will cover multiple profiles for end-to-end redundancy minimization in the next section.

It is to be noted that our goal is to provide a generic framework supporting all the basic functionality for adapting an SSM-compliant resource. The parcel-mapping rule presented here is a generic way in which a resource can be adapted and packaged so that different recipients (profiles) requiring possibly different subsets of parcels at different renditions based on their capabilities and preferences can extract suitable versions from the adapted bit-stream. However, depending on the complexity of the use case in which this may be used, it may be more convenient to add a layer on SSM that collects parcels in the descriptor, resource and outbound constraints, and controls invocation of the SSM adaptation engine.

5.2.2 Adaptation constraints

The adaptation constraint specification for each parcel in a profile can be either driven by *adaptation variables* or by the *structure*. The former, which is based on defined adaptation variables, is a more interesting case because it keeps the content creator and the receiver sides more independent of each other, and integrates decision making within an adaptation engine. However, if the receiver knew exactly the structure of the content, it could specify the adaptation points exactly in its constraint specification. This is called *structure driven* adaptation, and in this case, there is no decision making involved on part of the adaptation engine.

Adaptation variable driven constraints, consist of a set of *limit constraints,* followed by an optional *optimization constraint*. Each constraint, limit or optimization, is specified in terms of definable functions of adaptation variables, called an *adaptation metric*. The metric is defined using a stack expression involving adaptation variables corresponding to the parcel, as defined in the resource description metadata. The stack expression methodology used is the same as the one described in Section 4.4.3.

Limit constraints are specified as *lower and/or upper* limits of supportable value(s) for outbound connections for a defined metric. When both are specified we have effectively provided a range, and when both limits are the same, we have imposed an equality constraint. An example of a limit constraint is: *Codesize*/latency < 300 KB/s. Here *Codesize* is a feature variable, but 1/latency is specified in outbound constraints as a multiplier. Overall, this indicates a bandwidth restriction on received media. Another example is: *display_resolution*<800 diagonal pixels.

An optimization constraint specifies a requested *minimization* or *maximization* of a defined metric. An example of such a constraint is in rate-distortion optimization, where a metric such as *MeanSquaredError* + λ.*Codesize* is minimized. Here the *Codesize* variable corresponds to rate (R), while the *MeanSquaredError* variable corresponds to distortion (D). Encrypted domain transcoding based on packet truncation minimizing D+λ.R has been covered in prior art [28], [29].

Sometimes it may be necessary to maintain consistency of adaptation across parcels. For example, if each parcel is a GOP from a video sequence, we do not want to have different spatial resolutions for each of them. The outbound constraints XML incorporates a method to preserve limited dependencies across parcels by allowing references to adaptation variables from the previous parcel for the particular decision made for the previous parcel. However, because the outbound constraint part for the next parcel may not be known at the adaptation time for the current parcel, it is necessary to specify explicitly a list of adaptation variables to be remembered by the engine for use in adaptation of the next parcel. The values of these variables are computed based on the adaptation points decided for the current parcel. It is up to the resource description generator to provide relevant information to the outbound constraints generator to make sure that all *previous* references are correctly resolved.

5.2.3 Adapatation decisions made on constraints

The information contained in the resource descriptor XML and the outbound constraints XML specification is all that an adaptation engine needs to decide how to adapt each parcel. For adaptation variable driven adaptation request, when both a set of limit constraints and an optimization constraint are specified, the adaptation engine seeks the minimum or the maximum of the optimization metric, within the allowable adaptation point space carved by the limit constraints. For each parcel, the engine can in principle perform an exhaustive search over the joint space of all possible adaptation choices for all the constituent components. For each candidate decision point, the engine first evaluates the limit constraint metrics to see if they are satisfied. Limit constraints in both the outbound constraints specification (from the receiver side) as well as the resource descriptor (from the content creator side) are considered. If all limit constraints are satisfied, the single optimization metric is evaluated. The optimum decision point is one that not only satisfies all the limit constraints but also maximizes or minimizes the optimization metric over all cases also satisfying the limit constraints.

When only limit constraints are specified but no optimization constraint, there is no unique solution. Normally, the adaptation engine assumes a default optimization constraint. Note however, that the actual decision making algorithm is not a part of the proposed framework. The SSM framework involves ways to convey all the information needed to make decisions, and how the adaptation is to be conducted once decisions have been made. However, there is no restriction on the decision making process.

5.2.4 Unstructured and SSM type requests

While our primary motivation is to cater to SSM-model compliant content that maintains dependencies between atoms as discussed in Section 3.4, the methodology for specification of adaptation requests has been extended to cater to unstructured content too. Specifically, the adaptation request for each parcel in a profile can be either *SSM* type or *unstructured* type. In an unstructured type request, arbitrary adaptations not limited by the dependency structure imposed by the SSM model are allowed. Each atom is handled independently while ignoring the model dependencies. That is, an arbitrary subset of atoms from the data cube can be selected and included in the adapted resource bit-stream. In a SSM model based request, the atoms included for the parcel in the profile must conform to the dependency structure imposed by the SSM model, and a single adaptation point essentially determines the subset of atoms to be included, based on the exclusive/incremental information for each tier.

For a *SSM* type *adaptation variable driven* request, the combination of the limit constraints and an optional optimization constraint yields a single adaptation point that essentially determines the atoms to be included in each component, based on their model dependencies. On the contrary, for an *unstructured* type *adaptation varable drive*n request, all dependency information is ignored. If only limit constraints are specified and no optimization constraint, the adapted resource for each component consists of all atoms that satisfy the limit constraints. If there exists an optimization constraint, then there is only one solution yielding the maximum and the minimum, and therefore, only the atoms corresponding to the final solution in all components are included in the adapted resource.

For *structure driven* requests, the interpretation of certain XML elements that indicate a bounding box to include, depends on whether the request type is *SSM* or *unstructured*. For these elements, a point is specified to indicate a group of atoms to be included. For a *SSM* type *structure driven* request, the specified bounding box point is interpreted as a SSM decision point, and all atoms based on the exclusive/incremental information for tiers are included. On the contrary, for *unstructured* type *structure driven* requests, all

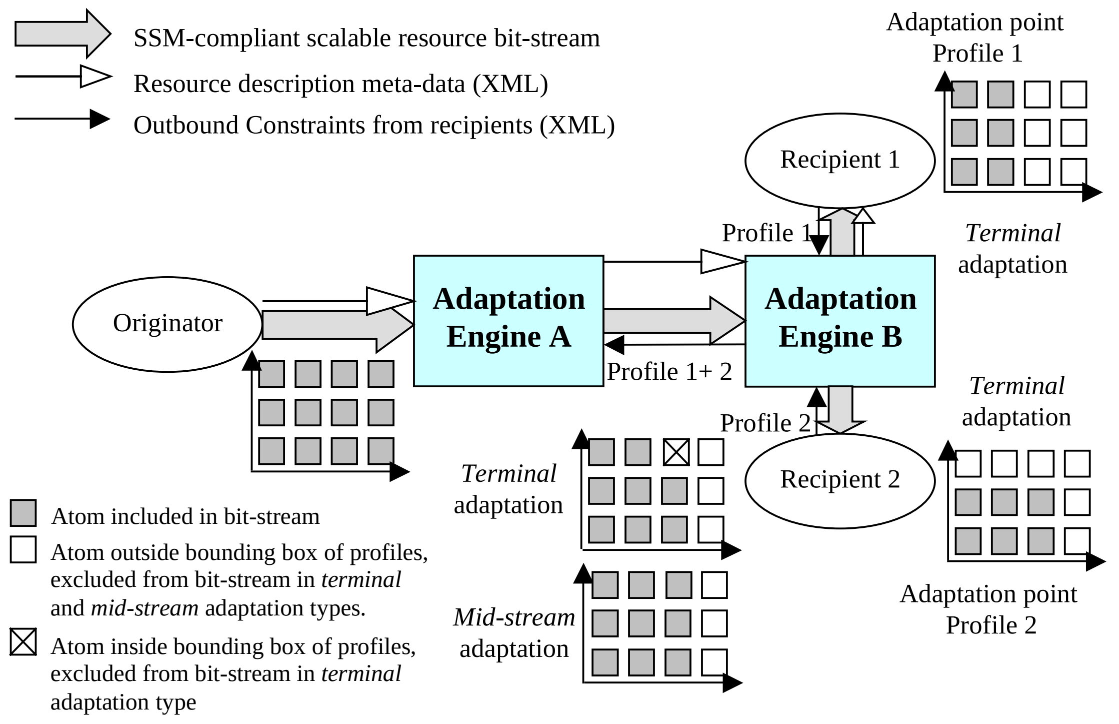

SSM-compliant scalable resource bit-stream
Resource description meta-data (XML)
Outbound Constraints from recipients (XML)
Adaptation point Profile 1
Recipient 1
Profile 1
Terminal adaptation
Originator
Adaptation Engine A
Adaptation Engine B
Profile 1+ 2
Profile 2
Terminal adaptation
Recipient 2
Terminal adaptation
Mid-stream adaptation
Adaptation point Profile 2
Atom included in bit-stream
Atom outside bounding box of profiles, excluded from bit-stream in terminal and mid-stream adaptation types.
Atom inside bounding box of profiles, excluded from bit-stream in terminal adaptation type

Figure 14. Multi-stage multi-profile adaptation example

atoms in the bounding box, irrespective of exclusive/incremental information for tiers are included.

6. Mid-stream Adaptation for multiple recipient profiles

If the outbound constraints XML contains exactly one receiver profile, a single adapted version of each SSM component allowing unambiguous decryption/decoding is generated and transmitted. This is typically the case when an adaptation engine directly connects to the eventual recipient.

There is however another scenario where an adaptation engine must adapt and deliver a bit-stream containing a combination of several profiles for several recipients, to be eventually extracted by other downstream adaptation engines. In this situation, the former adaptation engine could send the bounding box containing the different versions for different receiver profiles, for both exclusive and incremental tiers, which allows re-adaptation to lower terminal versions downstream. If all atoms within the bounding box including those that are not used by any of the profiles are sent, it is referred to as *mid-stream* adaptation type. Alternatively, atoms unused by all profiles may be dropped from the bit-stream, thereby saving bandwidth. This type of adaptation is referred to as *terminal*. The outbound constraints specification also conveys the type of adaptation desired. Further, when atoms within the bounding box of profiles are dropped as a consequence of *terminal* adaptation, this information is conveyed in the adapted resource description metadata for use by subsequent adaptation engines.

Consider Figure 14 that shows a delivery chain to two recipients through two adaptation engines, along with the example of a SSM component. Both recipients convey their profiles in their respective outbound constraints specifications to adaptation engine B. Engine B aggregates the two profiles into single outbound constraints specification and forwards it upstream to adaptation engine A, requesting either *mid-stream* or *terminal* adaptation types. The engine A actually decides on the adaptation points for each profile, and packs the bit-stream using either mid-stream or terminal adaptation type as requested by engine B, and forwards it to B. On receipt of the bit-stream, B applies the same profiles to the input stream but individually for each of the two output streams for the two recipients, performing *terminal* type adaptation in this case. Since the bit-stream transmitted by A already contains the optimal decisions for both recipients, the same optimization over a reduced set of possibilities would yield the same solutions individually. Hence, engine B is able to extract the bit-stream needed for each recipient, irrespective of the adaptation type *mid-stream* or *terminal*, from its input bit-stream, and deliver to the recipients. The only difference between the two adaptation types used by engine A is that the *mid-stream* type leaves more flexibility for engine B to create further versions other than those needed by the two recipients, while the *terminal* type attempts to minimize the end-to-end redundancy of transmission bandwidth.

An alternative architecture is based on the assumption that the resource descriptor is available at both adaptation engines before the resource is transmitted through either engine. In this case, the engine B could do the decision making itself for the two recipients, and send only a *structure-driven* adaptation request to engine A. The required bit-streams are packaged in the same way as in the previous case by engine A and sent back to B, which then uses single-profile *structure-driven terminal* adaptations to extract the appropriate bit-stream for each for delivery to each recipient.

Note that the SSM-enabled multi-step multi-profile architecture above, supports end-to-end adaptation and delivery of scalable content in a fully codec non-specific manner, while minimizing overall redundancy in bandwidth.

7. Schemas and semantics (Version 2.0)

In this section we describe the details of version 2.0 of the schemas.

7.1 Common XML Schema

Common XML schema includes the common XML schema data types and structures that are used by both the resource description XML schema and the outbound constraints XML schema.

Wrapper of the common XML schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns="SSMCommon" targetNamespace="SSMCommon"
        elementFormDefault="qualified" attributeFormDefault="unqualified">
```

Adaptation Variable Name Type

```
<xs:simpleType name="avarType">
        <xs:restriction base="xs:Name">
                <xs:pattern value="ssm:avar:\c+" />
        </xs:restriction>
</xs:simpleType>
```

Name	*Explanation*
avarType	All the adaptation variable names have to start with “ssm:avar:”. For example, the following are some valid names: *ssm:avar:imageResolution, ssm:avar:codesize*.

Component Name Type

```
<xs:simpleType name="compType">
        <xs:restriction base="xs:Name">
                <xs:pattern value="ssm:comp:\c+" />
        </xs:restriction>
</xs:simpleType>
```

Name	*Explanation*
compType	All the component names have to start with “ssm:comp:”. For example, the following are some valid component names: *ssm:comp:myImage*, *ssm:comp:myAudio*

Misc. Data Types

```
<xs:simpleType name="nonNegativeFloatType">
        <xs:restriction base="xs:float">
               <xs:minInclusive value="0" />
        </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nonNegativeIntegerListType">
        <xs:list itemType="xs:nonNegativeInteger" />
</xs:simpleType>

<xs:simpleType name="indexIntegerListType">
         <xs:list>
              <xs:simpleType>
                   <xs:restriction base="xs:integer">
                         <xs:minInclusive value="-1" />
                   </xs:restriction>
              </xs:simpleType>
          </xs:list>
  </xs:simpleType>

<xs:simpleType name="positiveIntegerListType">
        <xs:list itemType="xs:positiveInteger" />
</xs:simpleType>

<xs:simpleType name="nonNegativeFloatListType">
        <xs:list itemType="nonNegativeFloatType" />
</xs:simpleType>

<xs:simpleType name="floatListType">
        <xs:list itemType="xs:float" />
```

```
</xs:simpleType>
```

Name	*Explanation*
`nonNegativeFloatType`	Floating number type with value greater or equal to zero
`nonNegativeIntegerListType`	List of integer numbers with values greater or equal to zero
`indexIntegerListType`	List of integer numbers with values greater or equal to -1
`positiveIntegerListType`	List of integer numbers with values greater than zero
`nonNegativeFloatListType`	List of floating numbers with values greater or equals to zero
`floatListType`	List of floating numbers

Address Types

```
<xs:simpleType name="addressTypeEnum">
   <xs:restriction base="xs:token">
      <xs:enumeration value="relative" />
      <xs:enumeration value="absolute" />
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="endianTypeEnum">
  <xs:restriction base="xs:token">
    <xs:enumeration value="big" />
    <xs:enumeration value="small" />
  </xs:restriction>
</xs:simpleType>

<xs:attributeGroup name="attrGroupPosAdd">
   <xs:attribute name="start" type="xs:long" use="required" />
   <xs:attribute name="addressType" type="addressTypeEnum" default="absolute"  />
</xs:attributeGroup>

<xs:attributeGroup name="attrGroupPosAddLen">
   <xs:attributeGroup ref="attrGroupPosAdd" />
   <xs:attribute name="length" type="xs:unsignedLong" use="required" />
</xs:attributeGroup>

<xs:attributeGroup name="attrGroupPosAddLenBit">
   <xs:attributeGroup ref="attrGroupPosAddLen" />
   <xs:attribute name="bitPos" type="xs:unsignedByte" default="0" />
   <xs:attribute name="signed" type="xs:boolean" default="true" />
   <xs:attribute name="endian" type="endianTypeEnum" default="big" />
</xs:attributeGroup>
```

Name	*Explanation*
`addressTypeEnum`	Type used to indicate the address type. Possible values

	are: *relative* and *absolute*
`endianTypeEnum`	Type used to indicate the endian type for the value. Possible values are: *big* and *small*. *big* means big endian, and *small* means small endian.
`attrGroupPosAdd`	This attribute group includes two attributes: `start`: Mandatory attribute. Long integer indicates the starting address. `addressType`: Optional attribute with default value *absolute*. Use type `addressTypeEnum` described above. If the value is *absolute*, it indicates that the start address is absolute address. If the value is *relative*, it indicates that the start address is relative address.
`attrGroupPosAddLen`	This attribute group includes the attribute group `attrGroupPosAdd` described above, and plus one additional mandatory attribute: `length`: Unsigned long integer indicates the length of the segment in bits or bytes depending on the context.
`attrGroupPosAddLenBit`	This attribute group includes the attribute group `attrGroupPosAddLen` described above, and plus the following additional optional attributes: `bitPos`: Unsigned byte integer indicates the starting bit position of the address. Default value is *0*. If the value is *n*, it indicates that the address starts at the *n*-th bit of the starting address specified by the `start` attribute and the address spans the number of bits specified by the `length` attribute. It is assumed that the MSB (most significant bit) of a byte is bit 0 (*n* = 0), while the LSB (least significant bit) is bit 7 (*n* = 7). `signed`: Boolean type to indicate if the value that is stored at the address in the resource is a signed value or an unsigned value. Default value is *true*. If the value is a signed value and is less than 0, the value will be stored using 2's compliment. `endian`: Use type `endianTypeEnum` described earlier. If the attribute value is *big*, we will use big endian method for the value stored on the resource. If the attribute value is *small*, we will use small endian method for the value stored on the resource. Default is *big*.

Operation Types

```
<xs:simpleType name="operationType">
  <xs:restriction base="xs:token">
      <xs:enumeration value="inverse" />
      <xs:enumeration value="negative" />
```

```
        <xs:enumeration value="magnitude" />
        <xs:enumeration value="log" />
        <xs:enumeration value="log10" />
        <xs:enumeration value="exp" />
        <xs:enumeration value="power10" />
        <xs:enumeration value="sqr" />
        <xs:enumeration value="sqrt" />
        <xs:enumeration value="clampZ" />
        <xs:enumeration value="boolIsNZ" />
        <xs:enumeration value="boolIsLEZ" />
        <xs:enumeration value="boolIsGEZ" />
        <xs:enumeration value="boolNOT" />
       <xs:enumeration value="add" />
       <xs:enumeration value="subtract" />
       <xs:enumeration value="absdiff" />
       <xs:enumeration value="multiply" />
       <xs:enumeration value="divide" />
       <xs:enumeration value="maximum" />
       <xs:enumeration value="minimum" />
       <xs:enumeration value="average" />
       <xs:enumeration value="boolOR" />
       <xs:enumeration value="boolAND" />
       <xs:enumeration value="boolXOR" />
       <xs:enumeration value="selector" />
   </xs:restriction>
</xs:simpleType>
```

Name	*Explanation*
`operationType`	All the operations that are supported. The operands are taken by popping elements from an expression stack. The following operators take one operand v, obtained by popping the stack: *inverse*: for value v, the result is 1/v *negative*: for value v, the result is –v. *magnitude*: for a positive value v, v will be returned; for a negative value v, –v will be returned. *log*: for a value v, the result is the natural logarithm of v *log10*: for a value v, the result is the base-10 logarithm of v *exp*: for a value v, the result is the exponential value e^v. *power10*: for a value v, the result is10 raised to the power of v,10^v. *sqr*: for a value v, the result is v^2 *sqrt*: for a value v, the result is the square root of v. *clampZ*: for a positive value v, v will be returned; for a negative value v, 0 will be returned. *boolIsNZ*: for a value v, when v is not zero, the result is 1, otherwise the result is 0. *boolIsLEZ*: for a value v, when v is less than or equal to zero, the result is 1, otherwise the result is 0.

	boolIsGEZ: for a value v, when v is greater than or equal to zero, the result is 1, otherwise the result is 0. *boolNOT*: for a value v, when v is zero, the result is 1, otherwise, the result is 0. The following operators take two operands v0 and v1, obtained by popping the stack in order: *add*: for values v0 and v1, the result is v0+v1 *subtract*: for values v0 and v1, the result is v0-v1 *absdiff*: for values v0 and v1, the result is the absolute difference between v0 and v1. *multiply*: for values v0 and v1, the result is v0*v1 *divide*: for values v0 and v1, the result is v0/v1 *maximum*: for values v0 and v1, the result is the bigger one from v0 and v1. *minimum*: for values v0 and v1, the result is the smaller one from v0 and v1. *average*: for values v0 and v1, the result is the average between v0 and v1. *boolOR*: for values v0 and v1, the result is the logical OR of v0 and v1 *boolAND*: : for values v0 and v1, the result is the logical AND of v0 and v1 *boolXOR*: : for values v0 and v1, the result is the logical exclusive OR of v0 and v1 The following operators take three operands v0, v1, and v2, obtained by popping the stack in order: *selector*: for values, v0, v1, and v2, if v0 is not zero, the result is v1, otherwise, the result is v2

Adaptation/Component Variable And Stack Expression Types

```
<xs:complexType name="adapVarType">
        <xs:attribute name="avar" type="avarType" />
        <xs:attribute name="previous" type="xs:boolean" default="false" />
</xs:complexType>

<xs:complexType name="compVarType">
            <xs:attribute name="compID" type="compType" />
            <xs:attribute name="indType">
                        <xs:simpleType>
                                <xs:restriction base="xs:token">
                                        <xs:enumeration value="inclusionInd" />
                                        <xs:enumeration value="layerInd" />
                                        <xs:enumeration value="origLayersInd" />
                                        <xs:enumeration value="curLayersInd" />
                                </xs:restriction>
                        </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="param" type="xs:nonNegativeInteger" />
        <xs:attribute name="previous" type="xs:boolean" default="false" />
</xs:complexType>

<xs:complexType name="stackExpnType">
        <xs:sequence maxOccurs="unbounded">
              <xs:choice>
                    <xs:element name="adapVar" type="adapVarType" />
                    <xs:element name="compVar" type="compVarType" />
                    <xs:element name="constant">
                          <xs:complexType>
                              <xs:attribute name="value" type="xs:float" />
                          </xs:complexType>
                    </xs:element>
                    <xs:element name="argument">
                          <xs:complexType>
                              <xs:attribute name="number" type="xs:nonNegativeInteger" />
                          </xs:complexType>
                    </xs:element>
                    <xs:element name="operation">
                          <xs:complexType>
                              <xs:attribute name="operator" type="operationType" />
                          </xs:complexType>
                    </xs:element>
              </xs:choice>
        </xs:sequence>
</xs:complexType>
```

Name	*Explanation*
`adapVarType`	Adaptation variable type used in stack expressions. It includes the following attributes: `avar`: Mandatory attribute indicates the adaptation variable name. It uses `avarType` described earlier. `previous`: Optional boolean type attribute with default value equals to *false*. If the `previous` attribute is set to be *true*, it indicates that we would like to use the adaptation variable value from the previous parcel.
`compVarType`	Component derived variable type used in stack expressions. It has the following attributes: `compID:` Mandatory attribute indicates the component name. It uses `compType` described earlier. `indType`: Mandatory attribute has the following possible values: *inclusionInd*, *layerInd*, *origLayersInd*, *and curLayersInd*. When `indType` is set to be *inclusionInd*, attribute `param` is not used. If the component specified by `compID` is included in the outbound adapted resource (i.e. at least one of its atoms is included), the return value is 1; otherwise it returns 0. If the `indType` is set to be *layerInd*, attribute `param` is used to indicate the tier index. It will return the number of layers in this tier that are within the bounding

	box for an adaptation decision. If the `indType` is set to be *origLayersInd*, attribute `param` is used to indicate the tier index. It will return a constant that is equal to the original number of layers in this tier from the resource prior to all adaptation steps. If the `indType` is set to be *curLayersInd*, attribute `param` is used to indicate the tier index. It will return a constant that is equal to the current number of layers in this tier from the resource prior to the current adaptation step. `previous:` Optional boolean attribute with default value equals to *false*. If the `previous` attribute is set to be *true*, it indicates that we would like to use the component derived variable value from the previous parcel.
`stackExpnType`	Stack expression type could include unlimited number of the following elements: `adapVar`: Use type `adapVarType` described earlier, and it is the name of the adaptation variable. `compVar`: Use type `compVarType` described earlier, and it is the name of the component derived variable. `constant`: attribute `value` indicates a constant floating number `argument`: attribute `number` indicates the argument index number. Element `argument` is allowed in the stack expressions only while defining combination variables. `operation`: attribute `operator` uses type `operationType` and indicates the operation we want to perform. All the child elements of a stack expression will be processed sequentially from top to bottom. If the child is `adapVar` or `compVar`, the value of the variable will be calculated and pushed to a value stack. If the child element is `constant`, the value of the constant will be push to the value stack. If the child element is `argument`, the value of argument *n* indicated by the `number` attribute will be pushed to the value stack. If the child element is `operation`, depending on the value of the attribute `operator`, 1, 2, or 3 values will be popped from the value stack, and used as the operands with the operator, and the result value will be pushed to the value stack again. After all the child elements of a stack expression are processed, there should be one and only one value left in the value stack, and it would be the final value of the stack expression.

Limit Constraint Types

```
<xs:attributeGroup name="attrGroupLimits">
  <xs:attribute name="lowLimit" type="xs:float" default="1" />
  <xs:attribute name="highLimit" type="xs:float" default="1" />
</xs:attributeGroup>

<xs:complexType name="limitConstraintType">
```

```
    <xs:complexContent>
        <xs:extension base="stackExpnType">
            <xs:attributeGroup ref="attrGroupLimits" />
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Name	*Explanation*
`attrGroupLimits`	Attribute group used to define limit values. It includes the following optional attributes: `lowerLimit`: one floating number is used to define the lower limit. Default value is 1. `upperLimit`: one floating number is used to define the upper limit. Default value is 1.
`limitConstraintType`	Limit constraint type uses `stackExpnType` described earlier with additional attribute group `attrGroupLimits`. If the result value from the stack expression is *v*, then *v* has to be equal or greater than the value of the attribute `lowLImit`, and also *v* has to be equal or less than the value of the attribute `highLimit`.

Example 1

```
<limitConstraint lowLimit="400" highLimit="960">
    <ssm:adapVar avar="ssm:avar:imageResolution" />
</limitConstraint>
```

The above example shows a limit constraint that uses the value of adaptation variable *ssm:avar:imageResolution*. If the final value for a particular adaptation point is within the range of 400 and 960, the limit constraint will be satisfied, otherwise the adaptation point could not satisfy the limit constraint.

Example 2

```
<limitConstraint lowLimit="0" highLimit="0">
    <ssm:adapVar avar="ssm:avar:imageResolution" />
    <ssm:adapVar avar="ssm:avar:imageResolution" previous="true" />
    <ssm:operation operator="subtract" />
</limitConstraint>
```

The above example shows a limit constraint that ensures the value of adaptation variable *ssm:avar:imageResolution* for this parcel is the same as the value for the previous parcel.

Optimization Constraint Types

```
<xs:simpleType name="optimizeType">
    <xs:restriction base="xs:token">
        <xs:enumeration value="maximize" />
        <xs:enumeration value="minimize" />
    </xs:restriction>
```

```
</xs:simpleType>
<xs:complexType name="optimizationConstraintType">
   <xs:complexContent>
      <xs:extension base="stackExpnType">
         <xs:attribute name="optimize" type="optimizeType" use="required" />
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Name	*Explanation*
`optimizeType`	There are two possible values for the type: *maximize* and *minimize*
`optimizationConstraintType`	Optimization constraint type uses `stackExpnType` described earlier with one additional mandatory attribute `optimize` of type `optimizeType` to indicate if we want to maximize or minimize the value of the stack expression.

Example

```
<optimizationConstraint optimize="maximize">
 <ssm:constant value="-0.1" />
 <ssm:adapVar avar="ssm:avar:codesize" />
 <ssm:operation operator="multiply" />
 <ssm:adapVar avar="ssm:avar:perceptualRichness" />
 <ssm:operation operator="add" />
</optimizationConstraint>
```

The above example shows that we would like to maximize the value from the expression: ssm:avar:perceptualRichness + (ssm:avar:codesize * -0.1)

Adaptation Variable Store Type

```
<xs:complexType name="adapVarStoreType">
   <xs:sequence maxOccurs="unbounded">
      <xs:choice>
               <xs:element name="storedAdapVar" type="adapVarType" />
               <xs:element name="storedCompVar" type="compVarType" />
          </xs:choice>
     </xs:sequence>
</xs:complexType>
```

Name	*Explanation*
`adapVarStoreType`	This type is used to specify the adaptation variables or component derived variables that we would like to store their values for the next parcel. In other words, these variable values could be used when evaluating the constraints for the next parcel. It could have unlimited number of child elements. The possible child elements include:

	`storedAdapVar`: Adaptation variable that we would like to store its value for the next parcel. Use type `adapVarType` described earlier. `storedCompVar`: Component derived variable that we would like to store its value for the next parcel. Use type `compVarType` described earlier.

Example

```
<adapVarStore>
  <ssm:storedAdapVar avar="ssm:avar:imageResolution" />
</adapVarStore>
```

The above example shows that we would like to store the value of *ssm:avar: imageResolution* variable from this parcel, and the value could be used when we evaluate some expressions in the next parcel.

7.2 Resource description XML Schema

Wrapper of the resource description XML schema

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
    xmlns:ssm="SSMCommon" xmlns="SSMDescription"
    targetNamespace="SSMDescription" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:import namespace="SSMCommon" schemaLocation="SSMCommon.xsd"/>
```

Tier and Tier Element Information Types

```
<xs:simpleType name="exclusiveModelType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="single" />
    <xs:enumeration value="firstAlways" />
    <xs:enumeration value="lastAll" />
    <xs:enumeration value="firstAlwaysLastAll" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="tierElemInfoType">
  <xs:attribute name="exclusiveFlag" type="xs:boolean" default="false" />
  <xs:attribute name="exclusiveModel" type="exclusiveModelType" default="single" />
  <xs:attribute name="numLayers" type="xs:positiveInteger" use="required" />
  <xs:attribute name="origLayers" type="xs:positiveInteger" use="required" />
  <xs:attribute name="tier" type="xs:nonNegativeInteger" use="required"  />
</xs:complexType>

<xs:complexType name="compDescriptionDataType">
   <xs:sequence>
      …
      <xs:element name="tierInfo">
         <xs:complexType>
            <xs:sequence>
            <xs:element name="tierElemInfo" type="tierElemInfoType" maxOccurs="unbounded" />
          </xs:sequence>
          <xs:attribute name="numTiers" type="xs:positiveInteger" />
        </xs:complexType>
      </xs:element>
      …
    </xs:sequence>
    …
</xs:complexType>
```

Name	*Explanation*
`exclusiveModelType`	Type used to describe the behavior model of an exclusive tier. It has the following possible values: *single*: All layers in the exclusive tier will be treated independently. When a layer is included in the adapted outbound resource, no other layers need to be included. *firstAlways*: The first layer in the exclusive tier is always included in the adapted outbound resource when any layer is included. *lastAll*: If the last layer of the exclusive tier is included in the adapted outbound resource, all layers in the tier will be included. *firstAlwaysLastAll*: This model includes the models *firstAlways* and *lastAll* described above.
`tierElemInfoType`	Tier element information type is used to describe the information in one tier element. It has the following attributes: `exclusiveFlag`: optional. Default value is *false*. If the value is *true*, this tier is exclusive; if the value is *false*, this tier is incremental. `exclusiveModel`: optional attribute of type `exclusiveModelType` described above. Default value is single. This attribute is only used when the value of the attribute `exclusiveFlag` equals to *true*. `numLayers`: mandatory positive integer to indicate the number of layers in this tier in the current version. `origLayers`: mandatory positive integer to indicate the number of layers in this tier in the original unadapted version. `tier`: mandatory non-negative integer to indicate the tier index.
`tierInfo`	Element in the component to describe the tier information. It may have unlimited number of `tierElemInfo` elements with type `tierElemInfoType`. It also has a positive integer attribute `numTiers` to indicate the total number of tiers. The value of `numTiers` should be the same of the total number of `tierElemInfo` elements.

Example

```
<tierInfo numTiers="2">
            <tierElemInfo numLayers="4" origLayers="4" exclusiveFlag="false" tier="0" />
            <tierElemInfo numLayers="5" origLayers="5" exclusiveFlag="false" tier="1" />
</tierInfo>
```

The above example shows there are two tiers in the component, and the first tier has 4 layers, and the second tier has 5 layers. Both tiers have incremental layers.

Component Table of Content Types

```
<xs:complexType name="atomTocType">
   <xs:sequence>
      <xs:element name="atomTocEntry" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
               <xs:attributeGroup ref="ssm:attrGroupPosAddLen">
               <xs:attribute name="indices" type="ssm:indexIntegerListType" use="required" />
            </xs:complexContent>
         </xs:complexType>
      </xs:element>
      <xs:element name="atomRemovedFromBBox" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
             <xs:attribute name="indices" type="ssm:indexIntegerListType" use="required" />
          </xs:complexType>
      </xs:element>
   </xs:sequence>
   <xs:attributeGroup ref="ssm:attrGroupPosAdd" />
</xs:complexType>

<xs:complexType name="compDescriptionDataType">
   <xs:sequence>
      …
      <xs:element name="atomToc" type="atomTocType" />
      …
   </xs:sequence>
   …
</xs:complexType>
```

Name	*Explanation*
`atomTocType`	Component table of content type. It includes the attribute group `ssm:attrGroupPosAdd` to indicate the address of a reference for the table of content in the `start` attribute and its address type in the `addressType` attribute. If the address type is *relative*, the reference address of the table of content is relative to the reference address of the table of content in the previous component. If the component is the first one in the parcel, then its reference address is relative to that of the last component in the previous parcel. If the component is the first one in the parcel, and the parcel is the first one in the resource, then the reference address is relative to 0. `atomTocType` may have unlimited number (including zero) of child

	elements `atomTocEntry`. Each `atomTocEntry` describes one entry in the table of content and it includes the attribute group `ssm:attrGroupPosAddLen` and one additional mandatory attribute `indices`. If the address type of `atomTocEntry` is *relative*, it means that the starting address of the `atomTocEntry` is relative to the reference address of the table of content. The sole purpose of the reference address in `atomTOCType` is to provide the location of its constituent `atomTOCEntry` elements relative to it. Attribute `indices` is used to indicate the indices for the table of content entry and is a list of integers with values greater or equal to -1. The number of values in the list should be the same as the number of dimensions(tiers) in the component. It is possible that multiple `atomTocEntry` elements have the same indices, or some indices do not have any corresponding `atomTocEntry` element defined. When –1 is used as one or more of the indices in the indices integer list, it means that these coordinates are regarded as don't cares. The atom needs to be included in the adapted outbound resource when a regular atom with the same coordinates for the non-don't care coordinates, are included. For example, atom (i,j,–1) is included whenever atom (i,j,n) is included for all n>=0. `AtomTocType` may also have any number (including zero) of child element `atomRemovedFromBBox`, which has one mandatory attribute `indices`. The attribute is used to indicate the indices for the atom within the bounding box that is removed from the outbound adapted resource and is a list of integers with values greater or equal to –1.
`atomToc`	Element in the component of type `atomTocType` to describe the table of content. Each component has one such element.

Example

```
<atomToc start="0" addressType="absolute">
  <atomTocEntry indices="0 0" start="15000" addressType="relative"  length="512" />
  <atomTocEntry indices="0 1" start="15600" addressType="relative"  length="360" />
  <atomTocEntry indices="0 2" start="16347" addressType="relative"  length="2000" />
  <atomTocEntry indices="0 3" start="18400" addressType="relative"  length="2536" />
  <atomTocEntry indices="0 4" start="21506" addressType="relative"  length="3078" />
  <atomTocEntry indices="1 0" start="25000" addressType="relative"   length="1006" />
  <atomTocEntry indices="1 1" start="26120" addressType="relative"   length="1878" />
  <atomTocEntry indices="1 2" start="27303" addressType="relative"   length="2663" />
  <atomTocEntry indices="1 3" start="30000" addressType="relative"  length="3549" />
  <atomTocEntry indices="1 4" start="36000" addressType="relative"  length="4812" />
  <atomTocEntry indices="2 0" start="40904" addressType="relative"   length="1470" />
  <atomTocEntry indices="2 1" start="42655" addressType="relative"   length="2351" />
  <atomTocEntry indices="2 2" start="45101" addressType="relative"  length="3534" />
  <atomTocEntry indices="2 3" start="48709" addressType="relative"   length="4915" />
  <atomTocEntry indices="2 4" start="53810" addressType="relative"   length="6002" />
  <atomTocEntry indices="3 0" start="60000" addressType="relative"  length="2029" />
  <atomTocEntry indices="3 1" start="62029" addressType="relative"  length="3147" />
  <atomTocEntry indices="3 2" start="65200" addressType="relative"  length="4258" />
  <atomTocEntry indices="3 3" start="69458" addressType="relative"  length="5307" />
  <atomTocEntry indices="3 4" start="74819" addressType="relative"  length="7003" />
</atomToc>
```

The above example shows that the reference address of the table of content uses *absolute* address type. All the entries in the table of content use *relative* address type. The resource data chunk B(2,3) starts at 48709+0 = 48709 and has length of 4915 bytes. There is no atom that is removed within the bounding box.

Component Description Data Type

```
<xs:complexType name="compDescriptionDataType">
   <xs:sequence>
      <xs:element name="tierInfo">
         …
      </xs:element>
      <xs:element name="atomToc" type="atomTocType" />
   </xs:sequence>
   <xs:attribute name="compID" type="ssm:compType" use="required" />
</xs:complexType>
```

Name	*Explanation*
`compDescriptionDataType`	Component description data type. It has one mandatory attribute `compID` to indicate the component name. It has the following elements: `tierInfo`: Tier information described earlier. `atomToc`: Table of content information described earlier.

Example

```
<component compID="ssm:comp:myAudio">
   <tierInfo numTiers="1">
      <tierElemInfo numLayers="6" origLayers="6" exclusiveFlag="false" tier="0" />
   </tierInfo>
   <atomToc start="90000" addressType="relative">
      <atomTocEntry indices="0" start="1178" addressType="relative"  length="789" />
      <atomTocEntry indices="1" start="1967" addressType="relative"  length="1745" />
      <atomTocEntry indices="2" start="3825" addressType="relative"  length="2840" />
      <atomTocEntry indices="3" start="6778" addressType="relative"  length="4173" />
      <atomTocEntry indices="4" start="10951" addressType="relative"  length="5281" />
      <atomTocEntry indices="5" start="16378" addressType="relative"  length="6989" />
   </atomToc>
</component>
```

The above example shows the information of the component *ssm:comp:myAudio*. The component has one tier and the tier has 6 incremental layers. Its table of content reference is at address (90000 + the starting reference address of the table of content in the previous component). The table of content has 6 entries. For example, the atom B(4) starts at (

10951 + the reference address of the table of content of this component) and is 5281 bytes long.

Feature and Feature Distribution Types

```
<xs:complexType name="marginalDistType">
  <xs:simpleContent>
    <xs:extension base="ssm:nonNegativeFloatListType">
      <xs:attribute name="dims" type="xs:positiveInteger" />
      <xs:attribute name="dimToTierMap" type="ssm:nonNegativeIntegerListType" use="required"
/>
      <xs:attribute name="distType">
        <xs:simpleType>
          <xs:restriction base="xs:token">
            <xs:enumeration value="nonDecreasing" />
            <xs:enumeration value="nonIncreasing" />
            <xs:enumeration value="nonMonotonic" />
            <xs:enumeration value="constant" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="featureDistType">
  <xs:sequence>
    <xs:element name="marginalDist" type="marginalDistType" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="featureDataType">
  <xs:sequence>
    <xs:element name="components">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="component" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="compID" type="ssm:compType" />
              <xs:attribute name="numLayers" type="ssm:positiveIntegerListType" use="optional"
/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="refFeatureValue" type="ssm:nonNegativeFloatType" />
    <xs:element name="emptyFeatureDist" type="ssm:nonNegativeFloatType" />
    <xs:element name="featureDist" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
         <xs:complexContent>
           <xs:extension base="featureDistType">
             <xs:attribute name="emptyComponentTiers" type="ssm:nonNegativeIntegerListType"
default="" />
            </xs:extension>
         </xs:complexContent>
       </xs:complexType>
     </xs:element>
  </xs:sequence>
  <xs:attribute name="avar" type="ssm:avarType" use="required" />
</xs:complexType>
```

Name	*Explanation*
`marginalDistType`	Marginal distribution type has its content of type `nonNegativeFloatListType` with the following additional attributes: `dims`: positive integer to indicate the number of dimensions in the marginal distribution. `dimToTierMap`: Non-negative integer list to indicate the tier indices that are included in the marginal distribution. The number of non-negative integers in the list should be the same as the value for the attribute `dims`. `distType`: distribution type with the following possible values: *nonDecreasing*, *nonIncreasing*, *nonMonotonic*, *constant.* For a *constant* type distribution, it indicates that the marginal distribution will always contain a single constant floating number. The number of values in the floating-point list should be the same as the product of the dimensions in the included tiers, unless `distType` equals to *constant*, in which case there is only one. Further the scan order for the values assume that the last tier specified in the `dimToTierMap` attribute is traversed first (i.e. constitutes the innermost loop), and then the second last, and so on.
`featureDistType`	Feature distribution type could include unlimited `marginalDist` elements. `marginalDist` uses marginalDistType. Together however, they should cover all the tiers. That is, the sum of the values of the attribute `dims` in all `marginalDist` should be the same as the number of the total tiers from all the components used in the feature.
`components`	Element `components` could have unlimited number of child element `component`. Each `component` has the following attributes: `compID`: Used to specify the component name. The specified components are those over which the feature distribution is defined jointly. `numLayers`: Optional attribute. The value of the attribute is a list of positive integers to indicate the number of layers in the tiers of the component. For a global feature, if the attribute `numLayers` is not specified, the value of the attribute `origLayers` in the `tierElemInfo` element in the component is used to determine the number of layers in the compoent. For a feature within a parcel, if the attribute

	numLayers is not specified, the value of the attribute numLayers in the tierElemInfo element in the component is used to determine the number of layers in the component. For a feature within a parcel, if the numLayers attribute is specified, these values take precedence over those provided in the attribute numLayers in the tierElemInfo element in the component, and the interpretation of the distribution is based on the former. In this case however, the values in attribute numLayers should be at least as big as the values in attribute numLayers in the tierElemInfo element.
featureDataType	This type is used to specify feature adaptation variables that involve one or more than one components (product features). It has the following child elements: components: described above. The specified components are those over which the feature distribution is defined jointly. refFeatureValue: non-negative floating number to indicate the reference feature value. The feature distribution value will be multiplied by the reference feature value to become the final feature value. emptyFeatureDist: non-negative floating number to indicate the feature distribution value when none of the components is included in the outbound adapted resource, *i.e.* all atoms are dropped from all components. featureDist: An unlimited number of children elements featureDist could be specified. Each is of type featureDistType with one additional attribute emptyComponentTiers of type non-negative integer list attribute. The list indicates the tiers for components that are not included in the outbound adapted resource, and has a default value of “”. The semantics is that if the feature has N components specified under components element, the number of featureDist elements should be exactly 2^N-1, to cover all the cases where at least one component is included. The case when all components are excluded has been covered by the emptyFeatureDist element. The tier mapping for the product distribution is zero-based and obtained by concatenating the tiers for the components in the order in which they are specified. The tiers specified in the emptyComponentTiers attribute of each featureDist are based on this mapping, and should include all tiers of all the components that are considered empty for that distribution. The marginalDist elements used to specify each featureDist must together cover all its non-empty tiers.

	Element `featureDataType` also has the following mandatory attribute: `avar`: adaptation variable name

Example 1

```
<feature avar="ssm:avar:imageResolution">
   <components>
      <component compID="ssm:comp:myImage" />
   </components>
   <refFeatureValue>1.0</refFeatureValue>
   <emptyFeatureDist>0.0</emptyFeatureDist>
   <featureDist>
      <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">100 200 400
800</marginalDist>
      <marginalDist dims="1" dimToTierMap="1" distType="constant">1.0</marginalDist>
   </featureDist>
</feature>
```

The above example specifies an adaptation feature *ssm:avar:imageResolution*. The feature uses the component *ssm:comp:myImage*. The reference feature value is 1.0, and the all-empty feature distribution value is 0.0. If the component is not included in the outbound adapted resource, the final feature value will be 1.0 * 0.0 = **0**. If the component originally has two tiers, and there are 4 layers in the first tier and there are 5 layers in the second tier, and if we use (2,3) as the adaptation point, the feature will return 1.0 * 400 * 1.0 = **400** as the feature value.

Example 2

```
<feature avar="ssm:avar:imageCodesize">
  <components>
     <component compID="ssm:comp:myImage" />
   </components>
   <refFeatureValue>1.0</refFeatureValue>
   <emptyFeatureDist>0.0</emptyFeatureDist>
   <featureDist>
      <marginalDist dims="2" dimToTierMap="0 1" distType="nonDecreasing">120 240 350 400 580
240 300 420 510 625 315 456 600 720 800 378 526 720 823 999</marginalDist>
    </featureDist>
</feature>
```

The above example specifies an adaptation feature *ssm:avar:imageCodesize*. The feature uses the component *ssm:comp:myImage*. The reference feature value is 1.0, and the all-empty feature distribution value is 0.0. If the component is not included in the outbound adapted resource, the final feature value will be 1.0 * 0.0 = **0**. If the component originally has two tiers, with 4 layers in the first tier (tier 0) and 5 layers in the second tier (tier 1), and if we use (2,3) as the adaptation point, the feature will return 1.0 * 720 = **720** as the feature value. If we use (3,4) as the adaptation point, the feature will return 1.0 * 999 = **999** as the feature value.

Example 3

```
<feature avar="ssm:avar:perceptualRichness">
   <components>
      <component compID="ssm:comp:myImage" />
      <component compID="ssm:comp:myAudio" />
   </components>
   <refFeatureValue>10.0</refFeatureValue>
   <emptyFeatureDist>0.0</emptyFeatureDist>
   <featureDist emptyComponentTiers="0 1">
      <marginalDist dims="1" dimToTierMap="2">1 2 3 5 8 10</marginalDist>
   </featureDist>
   <featureDist emptyComponentTiers="2">
      <marginalDist dims="1" dimToTierMap="0">1 1.2 1.5 2</marginalDist>
      <marginalDist dims="1" dimToTierMap="1">1 2 3 4 4.5</marginalDist>
   </featureDist>
   <featureDist>
     <marginalDist dims="3" dimToTierMap="0 1 2">1 2 3 4 4 5 2 3 3 4 5 5 2 3 3 4 5 6 3 3 4 4 5 6 3 3 4 5
6 6 3 4 5 6 6 7 4 5 5 6 7 7 4 5 5 6 7 8 5 5 6 6 7 8 5 5 6 7 8 8 6 7 8 9 9 10 7 8 8 9 10 10 7 8 8 9 10 11 8 8 9 9
10 11 8 8 9 10 11 11 11 12 13 14 14 15 12 13 13 14 15 15 12 13 13 14 15 16 13 13 14 14 15 16 13 13 14
15 16 16</marginalDist>
   </featureDist>
</feature>
```

The above example specifies an adaptation feature *ssm:avar:perceptualRichness*. The product feature involves two components, *ssm:comp:myImage* and *ssm:comp:myAudio*. The reference feature value is 10.0, and the all-empty feature distribution value is 0.0. If neither one of the two components is included in the outbound adapted resource, the final feature value will be 10.0 * 0.0 = **0**. Let's assume that the component *ssm:comp:myImage* has two tiers, and originally there are 4 layers in the first tier and there are 5 layers in the second tier, and the component *ssm:comp:myAudio* has one tier, and there are originally 6 layers in that tier. If component *ssm:comp:myImage* is not included in the outbound adapted resource, we will use the first *featureDist* to calculate the feature value. If component *ssm:comp:myAudio* is not included in the outbound adapted resource, we will use the second *featureDist* to calculate the feature value. If both of them are included, we use the third *featureDist* to calculate the feature value.

Combination Adaptation Variable Type

```
<xs:complexType name="combAvarType">
   <xs:complexContent>
      <xs:extension base="ssm:stackExpnType">
         <xs:attribute name="avar" type="ssm:avarType" use="required" />
         <xs:attribute name="numArguments" type="xs:nonNegativeInteger" default="0" />
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Name	*Explanation*
`combAvarType`	Combination adaptation variable type is based on type `ssm:stackExpnType` with the following attributes: `avar`: Mandatory attribute used to indicate the name of the

	combination adaptation variable. `numArguments`: optional attribute. The non-negative number indicates the total number of arguments the combination adaptation variable uses. Default value is *0*. The combination adaptation variables used as the stored variables should take zero arguments.

Example 1

```
<globalCombAvar avar="ssm:avar:myImageImpliesAudio">
   <ssm:compVar compID="ssm:comp:myAudio" indType="inclusionInd" />
   <ssm:compVar compID="ssmc:comp:myImage" indType="inclusionInd" />
   <ssm:operation operator="boolNOT" />
   <ssm:operation operator="boolOR" />
</globalCombAvar>
```

The above example shows that the combination adaptation feature *ssm:avar: myImageImpliesAudio* has 4 child elements. This combination feature indicates that if component ssm:comp:myImage is included in the outbound adapted resource, component ssm:comp:myAudio has to be also included in the outbound adapted resource. If both component ssm:comp:myAudio and ssm:comp:myImage are included in the outbound adapted resource, the feature will return ((NOT true) OR (true) = **true** as the final feature value. If neither one of them is included, it will return ((NOT false) OR (false)) = **true** as the final return value. If ssm:comp:myAudio is included, but not ssm:comp:myImage, it will return ((NOT false) OR (true)) = **true** as the final return value. If ssm:comp:myImage is included, but not ssm:comp:myAudio, it will return ((NOT true) OR (false)) = **false** as the final return value.

Example 2

```
<globalCombAvar avar="ssm:avar:lagrangian" numArguments="2">
  <ssm:adapVar avar="ssm:avar:audioDistortion" />
  <ssm:adapVar avar="ssm:avar:imageDistortion" />
  <ssm:argument number="0" />
  <ssm:operation operator="multiply" />
  <ssm:operation operator="add" />
  <ssm:adapVar avar="ssm:avar:codesize" />
  <ssm:argument number="1" />
  <ssm:operation operator="multiply" />
  <ssm:operation operator="add" />
</globalCombAvar>
```

The above example shows that the combination adaptation variable *ssm:avar: lagrangian* uses 2 arguments. This combination adaptation variable will return the value from the following statement:

$(\text{argument}_1 * \textit{ssm:avar:codesize}) + \{(\text{argument}_0 * \textit{ssm:avar:imageDistortion}) + \textit{ssm:avar:audioDistortion})\}$

LUT (Look Up Table) Adaptation Variable Type

```
<xs:complexType name="LUTAvarType">
 <xs:sequence>
  <xs:element name="axisValues" maxOccurs="unbounded">
   <xs:complexType>
    <xs:attribute name="grid" type="ssm:floatListType" use="required" />
    <xs:attribute name="axis" type="xs:nonNegativeInteger" use="required" />
   </xs:complexType>
  </xs:element>
  <xs:element name="content">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="marginalDist" type="marginalDistType" maxOccurs="unbounded" />
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="numAxes" type="xs:positiveInteger" default="1" />
 <xs:attribute name="avar" type="ssm:avarType" use="required" />
 <xs:attribute name="interpolationMethod" default="linear">
  <xs:simpleType>
   <xs:restriction base="xs:token">
    <xs:enumeration value="linear" />
    <xs:enumeration value="round" />
    <xs:enumeration value="ceil" />
    <xs:enumeration value="floor" />
   </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
</xs:complexType>
```

Name	*Explanation*
`axisValues`	`axisValues` element is used to describe the axis values in the lookup table (LUT). It has the following mandatory attributes: `grid`: The values of the attribute is a list of floating numbers to indicate the grid values for the axis. The values in the list are in strictly ascending order. The total number of values in the list plus 2 determines the dimension of the grid. The dimension will be used to interpret the list in the LUT content. `axis`: Non-negative number to indicate the index of the axis.
`content`	Content in the LUT has unlimited number of child element `margianlDist` of type `marginalDistType` described earlier.
`LUTAvarType`	Lookup table adaptation variable type. It has any number of child element `axisValues` described earlier, and one child element `content` described earlier. It also has the following attributes: numAxes: positive integer to indicate the total number of axes in the LUT. Default value is 1. The number should be the same as the total number of child element `axisValues`. avar: Adaptation variable name. interpolationMethod: interpolation method that will be used when the

	input values fall between the grid values. Possible values are: *linear*, *round*, *ceil*, and *floor*. Default value is *linear*.

Example

```
<globalLUTAvar avar="ssm:avar:lagrangianLUT" numAxes="2">
 <axisValues axis="0" grid="25 50 100" />
 <axisValues axis="1" grid="10 30 100 300" />
 <content>
  <marginalDist dims="2" dimToTierMap="0 1">0.9 0.8 0.7 0.6 0.5 0.4 0.8 0.7 0.6 0.5 0.4 0.3 0.7 0.6 0.5
0.4 0.3 0.2 0.6 0.5 0.4 0.3 0.2 0.1 0.5 0.4 0.3 0.2 0.1 0.1</marginalDist>
 </content>
</globalLUTAvar>
```

The above example shows that the LUT adaptation variable *ssm:avar:lagrangianLUT* uses 2 axes. The first axis has 3 grid values and the dimension for it is 3+2=5. The second axis has 4 grid values and the dimension for it is 4+2=6. In the marginal distribution, there should be 5*6=30 values in the list. If the input arguments for the LUT adaptation variable are 25 and 10, the adaptation variable will return 0.7 as the value

Resource Edit Type

```
<xs:complexType name="resourceEditType">
  <xs:complexContent>
    <xs:extension base="ssm:stackExpnType">
      <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
      <xs:attribute name="outLength" use="optional">
        <xs:simpleType>
          <xs:union memberTypes="ssm:avarType xs:nonNegativeInteger" />
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Name	*Explanation*
`resourceEditType`	Resource edit type is used to specify how we would like to edit the resource. It is based on type `ssm:stackExpnType` explained earlier. The value of the stack expression will be used to update the resource. The value of the stack expression will be performed against the final adapted bounding box for the resource. The adaptation variables or the component variables in the stack expression should not have attribute `previous` set to *true*. It means that we can only use the adaptation variable values for this parcel to undate the resource, not the adaptation variable values from the previous parcel. `resourceEditType` has the following attribute group and attribute: `ssm:attrGroupPosAddLenBit`: This attribute group was explained earlier. It includes the starting address, address type,

	starting bit position, the total length in bits, its endian type, and whether it is signed or unsigned. If the address type is relative, it indicates that the starting address of the location where we would like to edit the resource is relative to the previous `resourceEdit` entry. If the entry is the first one in the parcel, then it's relative to the last `resourceEdit` in the previous parcel. If this is the first parcel, then it's relative to 0. `outLength`: optional attribute. If the attribute is not specified, it means that the length after editing the resource is the same as the length before the editing. The value of the `outLength` could be specified in two ways. One way is to use a non-negative integer to indicate the number of bits after the editing. If the value is set to 0, it means we would like to remove the resourceEdit entry from the resource. The other way is to use an adaptation variable to specify the attribute value. However, the stack expression in the adaptation variable should not have attribute `previous` set to *true*.

Example

```
<resourceEdit start="10000" addressType="absolute" length="8">
      <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="0" />
</resourceEdit>
```

The above example shows that we would like to store the component variable *layerInd* for tier 0 of component *ssm:comp:myImage* in the resource starting at address 10000 and the total length spanning 8 bits.

Offset Reference Data Type

```
<xs:simpleType name="invalidPointerHandlingType">
  <xs:restriction base="xs:token">
    <xs:enumeration value="moveUp" />
    <xs:enumeration value="moveDown" />
    <xs:enumeration value="zeroOut" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="offsetReferenceDataType">
  <xs:sequence>
    <xs:element name="offsetEntry" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="value" type="xs:long" use="required" />
          <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
          <xs:attribute name="invalidPointerHandling" type="invalidPointerHandlingType"
default="moveUp" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  <xs:attributeGroup ref="ssm:attrGroupPosAdd" />
  <xs:attribute name="invalidPointerHandling" type="invalidPointerHandlingType" default="moveUp"
/>
</xs:complexType>

<xs:element name="SSMDescription">
        …
        <xs:element name="codecOffsetData" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="offsetReference" type="offsetReferenceDataType" minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        …
</xs:element>
```

Name	*Explanation*
`invalidPointerHandlingType`	Type used to indicate how to handle a pointer that points to a dropped byte. There are three possible values: *moveUp*, *moveDown*, and *zeroOut*. *moveUp* means we will move the pointer to point to the next valid byte. *moveDown* means we will move the pointer to the previous valid byte. *zeroOut* means we will set the pointer to zero.
`offsetEntry`	Element used to describe one offset entry. Each `offsetEntry` uses attribute group `ssm:attrGroupPosAddLenBit` to specify the location and length in bits, the starting bit position of the entry, its endian type and its signed/unsigned type. If the value of attribute `addressType` in the attribute group is *relative*, it means that the starting address of the entry is relative to the starting address of the parent offset reference element. The offset value stored in the resource is always relative to the starting address of the starting address of the parent offset reference element. Each `offsetEntry` also has the following additional attributes: `value`: This attribute value will be the same value on the resource pointed by the `offsetEntry`. `invalidPointerHandling`. The attribute is used to specify how we would like to handle when the value points to a dropped byte. It uses type `invalidPointerHandlingType`

	described earlier. Default value is *moveUp*.
`offsetReferenceDataType`	Type used to describe the offset reference. Each may have any number of child element `offsetEntry` described earlier. Each `offsetReference` element uses attribute group `ssm:attrGroupPosAdd` to indicate the starting address of the `offsetReference`, and its address type using attributes `start` and `addressType` in the attribute group respectively. If the value of attribute `addressType` in the attribute group is *relative*, the starting address is relative to the starting address of the previous offset reference; if the offset reference is the first one, it is relative to 0. `invalidPointerHandling`: uses type `invalidPointerHandlingType` described earlier. If the offset reference points to a dropped byte, the attribute value will determine how we handle the invalid pointer. Default value is *moveUp*.
`codecOffsetData`	Used in resource description to describe codec offset data. It may have any number (including zero) of child element `offsetReference` with type `offsetReferenceDataType` described earlier. If there are no codec offset data to specify, the `codecOffsetData` element could be dropped.

Example

```
<codecOffsetData>
    <offsetReference start="85000" addressType="relative">
      <offsetEntry value="10000" start="2" length="16" addressType="relative"
invalidPointerHandling="moveUp" />
      <offsetEntry value="20000" start="4" length="16" addressType="relative"
invalidPointerHandling="moveUp" />
    </offsetReference>
    <offsetReference start="3000" addressType="relative">
      <offsetEntry value="-8000" start="2" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
      <offsetEntry value="30000" start="4" bitPos="4" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
     <offsetEntry value="50000" start="7" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
    </offsetReference>
</codecOffsetData>
```

The codec offset data has two offset references and the first one has two entries and the second one has three entries. The first offset reference starts at address 85000+0=85000, and the second offset reference starts at address 3000+85000=88000. The last offset entry in the second offset reference starts at address 7+88000=88007, and lasts for 20 bits. The value stored at address 88007 will be the relative address to 88000, and is treated as signed value.

Sequence Data Type

```
<xs:complexType name="sequenceDataType">
 <xs:sequence>
  <xs:choice maxOccurs="unbounded">
   <xs:element name="writeField" minOccurs="0">
    <xs:complexType>
     <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
     <xs:attribute name="type" default="count">
      <xs:simpleType>
       <xs:restriction base="xs:token">
        <xs:enumeration value="count" />
        <xs:enumeration value="seqValue" />
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
    </xs:complexType>
   </xs:element>
   <xs:element name="countField">
    <xs:complexType>
     <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
     <xs:attribute name="write" type="xs:boolean" default="true" />
    </xs:complexType>
   </xs:element>
   <xs:element name="subSequence" type="sequenceDataType" />
   <xs:element name="countOnly">
    <xs:complexType />
   </xs:element>
  </xs:choice>
 </xs:sequence>
 <xs:attribute name="startValue" type="xs:long" use="optional" default="0" />
 <xs:attribute name="stepValue" type="xs:long" use="optional" default="1" />
 <xs:attribute name="modulo" type="xs:long" use="optional"/>
 <xs:attribute name="pack" type="xs:boolean" use="optional" default="true" />
 <xs:attribute name="relativeStart" type="xs:boolean" use="optional" default="true" />
</xs:complexType>

<xs:element name="SSMDescription">
      …
      <xs:element name="sequencetData" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="sequence" type="sequenceDataType" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      …
</xs:element>
```

Name	*Explanation*
sequenceDataType	Type used to describe the sequence data. It can have the following attributes: startValue: Optional long integer with default value to be *0*. The start value of the sequence data. stepValue: Optional long integer with default value to be *1*. The step value of the sequence data. The sequence value will be incremented by the step value for each child element including countField, subSequence, and countOnly elements. modulo: Optional long integer to indicate the modulo value of the sequence. If the sequence value equals to or exceeds the modulo value, we will use the remainder (modulus) obtained by dividing the sequence value into the modulo value as the new sequence value. pack: Optional boolean attribute with default value to be *true*. If the value is *true*, when the child element (countField or subSequence) of the sequence data pointing to dropped field on the resource, the child element would be removed from the sequence data. If the value is *false*, when the child element (countField or subSequence) of the sequence data pointing to dropped field on the resource, the child element would be changed to countOnly element. relativeStart: Optional boolean attribute with default value to be *true*. The attribute is used when the sequenceDataType is used as a sub-sequence within a parent sequence data. If the value of the attribute relativeStart is *true*, the start value of the sequence data would be (the value of attribute startValue + the last sequence data of the parent sequence + the step value of the parent sequence). If the value of the attribute relativeStart is *false*, the start value of the sequence data would be the value of the attribute startValue. sequenceDataType can have unlimited number of the following child elements:

`writeField:` It is used to store the current count of the sequence or the current sequence value to the specific location on the resource. `writeField` uses attribute group `attrGroupPosAddLenBit` to indicate the location, length, and address type of the field where we would like to write the value to. If the address type is *relative*, it means the starting address is relative to the previous `countField` or `writeField` element or the last element in the previous `subSequence` element. If the `writeField` is the first one in the sequence, it means that the starting address of the field is relative to the last field in the parent sequence. If there is no parent sequence, it means that the starting address is relative to 0. The attribute `type` is an optional attribute with default value to be *count*. If the value of the attribute `type` is *count*, we would write the current count of child elements up till this `writeField` including `countField`, `subSequence`, and `countOnly` elements. If the value of the attribute `type` is *seqValue*, we will write the current sequence value up till this `writeField` to the resource. However, the sequence value will not be incremented for this `writeField`.

`countField`: Indicate a count field in the sequence. It uses attribute group `attrGroupPosAddLenBit` to indicate the location, length, and the address type of the field where we would like to write the sequence data to. If the address type is *relative*, it means the starting address is relative to the previous `countField` or `writeField` element or the last element in the previous `subSequence` element. If the `countField` is the first one in the sequence, it means that the starting address of the field is relative to the last field in the parent sequence. If there is no parent sequence, it means that the starting address is relative to 0. The attribute `write` is an optional boolean attribute with default value to be *true*. If the value of attribute `write` is *false*, we would not write the

	sequence data to the resource, we will still increment the sequence data by the step value for this `countField` element. `subSequence`: Indicate a sub-sequence within the sequence data. Uses type `sequenceDataType`. We will increment the sequence data by the step value for this `subSequence` element. `countOnly`: Indicate a count only field. It could be the result of a dropped field. No sequence data will be written to the resource; we will still increment the sequence data by the step value for this `countOnly` element.
`sequenceData`	Used in resource description to describe sequence data. It may have any number (including zero) of child element `sequence` with type `sequenceDataType` described earlier. If there are no sequence data to specify, the sequenceData element could be dropped.

Example1

```
<sequence startValue="10" stepValue="2">
  <countField start="10000" length="16" addressType="absolute" />
  <countField start="5000" length="16" addressType="relative" />
  <countField start="1000" length="16" addressType="relative" />
  <countField start="2000" length="16" addressType="relative" />
</sequence>
```

This example will write value **10** to address 10000, value **12** to address 15000, value **14** to address 16000, and value **16** to address 18000. All fields are two bytes long in the resource.

Example2

```
<sequence startValue="0" stepValue="1">
  <countField start="20000" length="8" addressType="absolute" write="false" />
  <countField start="6000" length="8" addressType="relative" write="false" />
  <countField start="4000" length="8" addressType="relative" write="false" />
  <countField start="3000" length="8" addressType="relative" write="false" />
  <writeField start="2000" length="8" addressType="relative" />
  <countField start="4000" length="8" addressType="relative" write="false" />
  <countField start="8000" length="8" addressType="relative" write="false" />
</sequence>
```

This example will write value **4** to address 35000 for the `writeField` element, since there are 4 `countField` elements in the sequence till this `writeField` element. The field is one byte long. No other fields will be updated since all the `countField` elements have “`write`” attribute value to be *false*.

Example3

```
<sequence startValue="0" stepValue="6">
 <writeField start="60000" length="8" addressType="absolute" type="seqValue" />
 <subSequence startValue="0" stepValue="1" pack="false">
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
 </subSequence>
 <subSequence startValue="0" stepValue="1" pack="false" relativeStart="false">
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countOnly />
  <countOnly />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countOnly />
  <countOnly />
 </subSequence>
 <subSequence startValue="0" stepValue="1" pack="false" modulo="8">
  <countField start="60000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countField start="1000" length="8" addressType="relative" write="true" />
  <countOnly />
  <countOnly />
  <countOnly />
  <writeField start="4000" length="8" addressType="relative" type="count" />
 </subSequence>
</sequence>
```

In this example, we will write value **0** to address 60000 for the `writeField` element. The field is one byte long.

For the first sub-sequence, we will write value 0+0=**0** to address 61000, value **1** to address 62000, value **2** to address 63000, …, value **5** to address 66000. All fields are one byte long.

For the second sub-sequence, since the `relativeStart` attribute value is *false*, we will write value **0** to address 67000, and value **3** to address 68000. All fields are one byte long.

For the third sub-sequence, since the `relativeStart` attribute value uses the default value *true*, we will write value (12+0) mod 8 = **4** to address 128000, value **5** to address 129000, and value **6** to address 130000. All fields are one byte long. For the writeField element, since there are 6 child elements including `countField` and `countOnly` elements up till this `writeField`, we will write value **6** to address 134000.

For all three sub-sequences, since the `pack` attribute values are all *false*, if any `countField` is dropped from adaptation in the sub-sequence, the `countField` will be changed to `countOnly` field. However, since the parent sequence uses `pack` attribute default value *true*, if all `countField` elements are dropped in the sub-sequence, the sub-sequence will be removed from the sequence.

SSMDescription

```
<xs:element name="SSMDescription">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="parcelData">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="parcel" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:element name="componentData">
                              <xs:complexType>
                                 <xs:sequence>
                                    <xs:element name="component" type="compDescriptionDataType"
maxOccurs="unbounded" />
                                 </xs:sequence>
                              </xs:complexType>
                              <xs:unique name="compID">
                                 <xs:selector xpath="component" />
                                 <xs:field xpath="@compID" />
                              </xs:unique>
                           </xs:element>

                           <xs:element name="featureData" minOccurs="0">
                              <xs:complexType>
                                 <xs:sequence>
                                    <xs:element name="feature" type="featureDataType" minOccurs="0"
maxOccurs="unbounded" />
                                  </xs:sequence>
                               </xs:complexType>
                            </xs:element>

                            <xs:element name="combAvarData" minOccurs="0">
                              <xs:complexType>
                                 <xs:sequence>
                                    <xs:element name="combAvar" type="combAvarType" minOccurs="0"
maxOccurs="unbounded" />
                                  </xs:sequence>
                               </xs:complexType>
                            </xs:element>

                            <xs:element name="LUTAvarData" minOccurs="0">
                             <xs:complexType>
                              <xs:sequence>
                               <xs:element name="LUTAvar" type="LUTAvarType" minOccurs="0"
maxOccurs="unbounded" />
                              </xs:sequence>
```

```
                                       </xs:complexType>
                                     </xs:element>

                                     <xs:element name="creatorLimitConstraints" minOccurs="0">
                                       <xs:complexType>
                                         <xs:sequence>
                                           <xs:element name="limitConstraint" type="ssm:limitConstraintType"
minOccurs="0" maxOccurs="unbounded" />
                                         </xs:sequence>
                                       </xs:complexType>
                                     </xs:element>

                                     <xs:element name="adapVarStore" type="ssm:adapVarStoreType"
minOccurs="0" maxOccurs="1" />

                                     <xs:element name="resourceEditData" minOccurs="0">
                                       <xs:complexType>
                                         <xs:sequence>
                                           <xs:element name="resourceEdit" type="resourceEditType" minOccurs="0"
maxOccurs="unbounded" />
                                         </xs:sequence>
                                       </xs:complexType>
                                     </xs:element>
                                   </xs:sequence>
                                   <xs:attribute name="parcelID" type="xs:unsignedLong" use="optional" />
                                 </xs:complexType>
                                 <xs:unique name="avar">
                                   <xs:selector
xpath="componentData/component/featureData/feature|combAvarData/combAvar|featureData/featu
re|LUTAvarData/LUTAvar" />
                                   <xs:field xpath="@avar" />
                                </xs:unique>
                              </xs:element>
                            </xs:sequence>
                          </xs:complexType>
                          <xs:unique name="parcelID">
                            <xs:selector xpath="parcel" />
                              <xs:field xpath="@parcelID" />
                            </xs:unique>
                          </xs:element>

            <xs:element name="globalFeatureData" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="globalFeature" type="featureDataType" minOccurs="0"
maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>

            <xs:element name="globalCombAvarData" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="globalCombAvar" type="combAvarType" minOccurs="0"
maxOccurs="unbounded" />
                    </xs:sequence>
```

```
        </xs:complexType>
      </xs:element>

      <xs:element name="globalLUTAvarData" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="globalLUTAvar" type="LUTAvarType" minOccurs="0"
maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="globalCreatorLimitConstraints" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="globalLimitConstraint" type="ssm:limitConstraintType"
minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="codecOffsetData" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="offsetReference" type="offsetReferenceDataType" minOccurs="0"
maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

      <xs:element name="sequencetData" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="sequence" type="sequenceDataType" minOccurs="0"
maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>

    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Name	*Explanation*
`parcel`	Parcel element defines one parcel. A parcel has the following elements: `componentData`: This element my have unlimited number of child element `component`, and each `component` is of type `compDescriptionDataType` described earlier. All the component names within one parcel have to be unique. `featureData`: This element may have any

number (including zero) of child element `feature`, and each `feature` is of type `featureDataType` described earlier. The feature info adaptation variables defined here apply to this parcel only. If there are no features to specify, the `featureData` element can be dropped.

`combAvarData`: This element may have any number (including zero) of child element `combAvar`, and each `combAvar` is of type `combAvarType` described earlier. The combination adaptation variables defined here apply to this parcel only. If there are no combination adaptation variables to specify, the `combAvarData` element can be dropped.

`LUTAvarData`: This element may have any number (including zero) of child element `LUTAvar`, and each `LUTAvar` is of type `LUTAvarType` described earlier. The LUT adaptation variables defined here apply to this parcel only. If there are no LUT adaptation variables to specify, the `LUTAvarData` element can be dropped.

All the adaptation variable names in one parcel including component feature adaptation variables, product component feature adaptation variables, combination adaptation variables, and LUT aadaptation variables have to be unique.

`creatorLimitConstraints`: This element may have any number (including zero) of child element `limitConstraint`, and each `limitConstraint` is of type `limitConstraintType` described earlier. All the limit constraints specified here need to be satisfied for this parcel when trying to find the optimized bounding box for the resource. If there is no creator limit constraint to specify, the `creatorLimitConstraints` element can be dropped.

`adapVarStore`: Each parcel can have at most one `adapvarStore` child element.

	adapVarStore is of type adapVarStoreType described earlier. The values of the adaptation variables or component variables will be sotred for possible later use by the next parcel. resourceEditData: This element may have any number (including zero) of child element resourceEdit, and each resourceEdit is of type resourceEditType described earlier. If there are no resourceEdit to specify here, the resourceEditData element can be dropped. The element specifies the resource modification we would like to perform for this parcel. Parcel element also has one optional unsigned long attribute parcelID. All the parcels in the description xml file must have unique parcelID values that increase sequentially. When a parcel does not have a parcelID attribute, it is assigned a default parcelID value equal to one greater than that of the previous parcel. If the first parcel in a profile does not have a parcelID attribute, it is assumed to have a default value of 0.
parcelData	parcelData may have unlimited number of child elements parcel described earlier.
globalFeatureData	Global feature adaptation variable data may have any number (including zero) of child element globalFeature, and each globalFeature is of type featuerInfoType described earlier. These global feature info adaptation variables defined here apply to all parcels. If there are no global features to specify the globalFeatureData element could be dropped. Since the global features apply to all parcels, the numbers of dimensions in the components as well as the numbers of tiers in the original resource have to be the same in all the parcels. The marginal distribution data in the global feature will not change after the adaptation. The distribution data maps to the number of tiers in the original

	resource.
`globalCombAvarData`	Global combination adaptation variable data may have any number (including zero) of child element `globalCombAvar`, and each `globalCombAvar` is of type `combAvarType` described earlier. These global combination adaptation variables defined here apply to all parcels. If there are no global combination adaptation variables to specify the `globalCombAvarData` element could be dropped
`globalLUTAvarData`	Global LUT adaptation variable data may have any number (including zero) of child element `globalLUTAvar`, and each `globalLUTAvar` is of type `LUTAvarType` described earlier. These global LUT adaptation variables defined here apply to all parcels. If there are no global LUT adaptation variables to specify the `globalLUTAvarData` element could be dropped
`globalCreatorLimitConstraints`	Global creator limit constraints may have any number (including zero) of child element `limitConstraint`, and each `limitConstraint` is of type `limitConstraintType` described earlier. All the limit constraints specified here need to be satisfied for all parcels when trying to find the optimized bounding bo for the resource. If there is no globl creator limit constraint to specify, the `creatorLimitConstraints` element can be dropped. Since global limit constraints apply to all parcels including the first parcel, we cannot set attribute `previous` to *true* in the stack expression of the limit constraints.
`SSMDescription`	Root element for the resource description XML. It has the following child elements described earlier: `parcelData`, `globalFeatureData`, `globalCombFeatureData`, `globalCreatorLimitConstraints`, `codecOffsetData`, and `sequenceData`.

Example

```
<SSMDescription xsi:schemaLocation="SSMDescription SSMDescription.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="SSMDescription"
xmlns:ssm="SSMCommon">
  <parcelData>
    <parcel parcelID="0">
      <componentData>
        <component compID="ssm:comp:myImage">
         …
        </component>
        <component compID="ssm:comp:myAudio">
        …
        </component>
      </componentData>
      <featureData>
      …
      </featureData>
      <combAvarData />
      <resourceEditData>
       …
      </resourceEditData>
    </parcel>
    <parcel parcelID="1">
      <componentData>
        <component compID="ssm:comp:myImage">
         …
        </component>
        <component compID="ssm:comp:myAudio">
        …
        </component>
      </componentData>
      <featureData>
      …
      </featureData>
      <combAvarData />
      <resourceEditData>
       …
      </resourceEditData>
    </parcel>
  </parcelData>
  <globalCombAvarData>
    …
  </globalCombAvarData>
  <globalLUTAvarData>
    …
  </globalLUTAvarData>
  <globalCreatorLimitConstraints>
   <globalLimitConstraint lowLimit="1" highLimit="1">
    <ssm:adapVar avar="ssm:avar:myImageImpliesAudio" />
   </globalLimitConstraint>
  </globalCreatorLimitConstraints>
  <codecOffsetData>
  …
  </codecOffsetData>
 <sequenceData>
```

```
    …
    </sequenceData>
</SSMDescription>
```

The above example shows that the resource description includes two parcels, and each parcel has two components. There is one global creator limit constraint define, and it requires that the feature value of *ssm:avar:myImageImpliesAudio* equals **1** for all parcels.

7.3 Outbound constraints XML schema

Wrapper of the outbound constraints XML schema

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
    xmlns:ssm="SSMCommon" xmlns="SSMAdapReq"
    targetNamespace="SSMAdapReq" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
  <xs:import namespace="SSMCommon" schemaLocation="SSMCommon.xsd"/>
```

Adaptation Variable Name Type

```
<xs:simpleType name="profileType">
  <xs:restriction base="xs:Name">
    <xs:pattern value="ssm:prof:\c+" />
  </xs:restriction>
</xs:simpleType>
```

Name	*Explanation*
`profileType`	All the profilee names have to start with "ssm:prof:". For example, the following are some valid names: *ssm:prof:terminal-1*, *ssm:prof:terminal-2*.

Adaptation Types

```
<xs:complexType name="adapVarDrivenAdaptType">
    <xs:sequence>
        <xs:element name="limitConstraint" type="ssm:limitConstraintType" minOccurs="0"
maxOccurs="unbounded" />
        <xs:element name="optimizationConstraint" type="ssm:optimizationConstraintType"
minOccurs="0" maxOccurs="1" />
    </xs:sequence>
</xs:complexType>

<xs:complexType name="structureDrivenAdaptType">
    <xs:sequence>
        <xs:element name="adaptationPoint" maxOccurs="unbounded">
            <xs:complexType>
                <xs:attribute name="layers" type="ssm:nonNegativeIntegerListType" use="required" />
                <xs:attribute name="compID" type="ssm:compType" use="required" />
                <xs:attribute name="incType" default="bboxInc">
                    <xs:simpleType>
                        <xs:restriction base="xs:token">
                            <xs:enumeration value="bboxDrop" />
                    <xs:enumeration value="bboxInc" />
                    <xs:enumeration value="atomInc" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
    </xs:sequence>
</xs:complexType>
```

Name	*Explanation*
`adapVarDrivenAdapType`	Feature driven adaptation type may have any number (including zero) of child element `limitConstraint` with type `limitConstraintType` described earlier. It may also have zero or one child element `optimizationConstraint` with type `optimizationConstratintType` described earlier.
`structureDrivenAdapType`	Structure driven adaptation type may have unlimited child elements `adaptationPoint`. Each `adaptationPoint` has the following attributes: `layers`: Non-negative integer list. The number of values in the list should be the same as the number of dimensions (tiers) in the component specified by the attribute `compID`. `compID`: Mandatory attribute to indicate the component name which this adaptation point is for. `incType`: optional attribute with possible value: *bboxDrop*, *bboxInc*, and *atomInc*. Default value is *bboxInc*. If the value is *bboxInc*, the values of the attribute `layers` are the numbers of layers we would like to keep in all the tiers. If the value is *bboxDrop*, the values of the attribute `layers` are the numbers of layers we would like to drop in all the tiers starting from the original number of layers. If the value is *atomInc*, we will include the one atom specified by the values of the attribute `layers`.

Example 1

```
<adapVarDriven>
    <limitConstraint>
    …
    </limitConstraint>
    <limitConstraint>
    …
    </limitConstraint>
    <optimizationConstraint optimize="minimize">
```

```
   …
   </optimizationConstraint>
</adapVarDriven>
```

The above example shows that the feature driven adaptation has two limit constraints and one optimization constraint.

Example 2

```
<structureDriven>
  <adaptationPoint compID="ssm:comp:myImage" layers="2 2" incType="bboxDrop" />
  <adaptationPoint compID="ssm:comp:myAudio" layers="5" incType="bboxInc" />
</structureDriven>
```

The above example shows that the structure driven adaptation has two adaptation points. The first one is for component *ssm:comp:myImage*, and we want to drop 2 layers in the first tier and 2 layers from the second tier. The second one is for component *ssm:comp:myAudio*, and we want to keep 5 layers in the first tier.

SSMAdapReq

```
<xs:element name="SSMAdaptReq">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="profile" maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="parcel" maxOccurs="unbounded">
                     <xs:complexType>
                        <xs:sequence>
                           <xs:choice>
                              <xs:element name="adapVarDriven" type="adapVarDrivenAdaptType" />
                             <xs:element name="structureDriven" type="structureDrivenAdaptType" />
                           </xs:choice>
                           <xs:element name="adapVarStore" type="ssm:adapVarStoreType"
minOccurs="0" maxOccurs="1" />
                        </xs:sequence>
                        <xs:attribute name="parcelID" type="xs:unsignedLong" use="optional" />
                        <xs:attribute name="requestType" default="SSM">
                          <xs:simpleType>
                            <xs:restriction base="xs:token">
                              <xs:enumeration value="SSM" />
                              <xs:enumeration value="unstructured" />
                              <xs:enumeration value="stopProfile" />
                              <xs:enumeration value="asIs" />
                            </xs:restriction>
                          </xs:simpleType>
                        </xs:attribute>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
               <xs:attribute name="profileID" type="profileType" />
            </xs:complexType>
            <xs:unique name="parcelID">
```

```
                <xs:selector xpath="parcel" />
                <xs:field xpath="@parcelID" />
              </xs:unique>
            </xs:element>
          </xs:sequence>
          <xs:attribute name="adaptType" default="terminal">
            <xs:simpleType>
              <xs:restriction base="xs:token">
                <xs:enumeration value="terminal" />
                <xs:enumeration value="midstream" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
        <xs:unique name="profileID">
          <xs:selector xpath="profile" />
          <xs:field xpath="@profileID" />
        </xs:unique>
      </xs:element>
```

Name	*Explanation*
`parcel`	Parcel may have one child element of either `adapVarDriven` or `structureDriven`. `adapVarDriven` is of type `adapVarDrivenType` described earlier, and `structureDriven` is of type `structureDrivenType` described earlier. Parcel also has either zero or one child element `adapVarStore` with type `adapVarStoreType` described earlier. It is used to specify the values of the adaptation variables or component variables we would like to store for possible future use in the next parcel. Parcel has an optional attribute `requestType` with the following possible values: *SSM*, *unstructured*, *stopProfile*, and *asIs*. The default value is *SSM*. When the parcel constraints are specified as adaptation variable driven, and `requestType` is *SSM*, one optimal bounding box will be chosen using the optimization constraint. If there is no optimization constraint specified, an implicit optimization constraint will be used to find the optimal bounding box. The implicit optimization will try to include as many layers as possible in each tier. When the parcel constraints are specified as adaptation variable driven, and `requestType` is *unstructured*; if there is an optimization constaint, only one optimal atom will be chosen. If there is no optimization constraint specified, all the atoms that fulfill the limit constraints will be chosen. Attribute value *stopProfile* means that the parcel will not be included in the adapted resource. Attribute value *asIs* means that the parcel will be included in the adapted resource as is. If the attribute value is *stopProfile* or *asIs*, the parcel cannot have `adapVarDriven` or `structureDriven` child element in it. Each parcel has one optional unsigned long attribute `parcelID`. All the values of `parcelID` in one profile have to be unique. When a

	parcel does not have a `parcelID` attribute, it is assigned a default `parcelID` value equal to one greater than that of the previous parcel. If the first parcel in a profile does not have a `parcelID` attribute, it is assumed to have a default value of 0.
`profile`	Profile may have unlimited child element `parcel` described earlier. Profile has an optional attribute `profileID` with type `profileType` to indicate the name of the profile. All the profile names have to be unique in the outbound constraints file.
`SSMAdapReq`	Root element for the outbound constraint XML. It has one optional attribute `adapType`. The possible values for `adapType` are *terminal*, and *midstream*. When the `adapType` is *terminal*, the atoms within the bounding box could be dropped if they are not needed for any of the profile. The default value is *terminal*. `SSMAdapReq` may have unlimited number of child element `profile` described earlier.

Example

```
<SSMAdaptReq adaptType="terminal" xsi:schemaLocation="SSMAdaptReq
SSMAdaptReq.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="SSMAdaptReq" xmlns:ssm="SSMCommon">
   <profile profileID="ssm:prof:terminal-1">
      <parcel parcelID="0">
         <adapVarDriven>
         …
          </adapVarDriven>
         <adapVarStore>
            <ssm:storedAdapVar avar="ssm:avar:imageResolution" />
         </adapVarStore>
      </parcel>
      <parcel parcelID="1">
         <adapVarDriven>
         …
         </adapVarDriven>
         <adapVarStore>
            <ssm:storedAdapVar avar="ssm:avar:imageResolution" />
         </adapVarStore>
      </parcel>
   </profile>
   <profile profileID="ssm:prof:terminal-2">
      <parcel parcelID="0">
         <structureDriven>
         …
         </structureDriven>
      </parcel>
      <parcel parcelID="1">
         <structureDriven>
         …
         </structureDriven>
      </parcel>
   </profile>
</SSMAdaptReq>
```

The above example shows an outbound constraints XML with adaptation type *terminal*, and it includes two profiles. The first profile has two parcels. Both of the parcels use adaptation variable driven adaptation, and store the value for one adaptation variable *ssm:avar:imageResolution* for the next parcel. The second profile has two parcels and the parcels use structure driven adaptation, and stores no features. All the parcels use the default value *SSM* for `requestType`.

8. Conclusion

Use of scalable media for content-agnostic adaptation is well known in the literature. These adaptation engines do not need to decrypt or decode compressed content in order to adapt it into a form appropriate for lower bandwidth/resolution etc. The underlying assumption behind the adaptation operation is that an engine understands the format in which the data is represented in, even though it does not need to know what the data actually is. However, the requirement on the structure of the content is still rigid in these approaches, because codec specific components are still needed for different types of media content. That is, an adaptation engine for images compressed in a particular way, say JPEG2000, would still be different from an adaptation engine for a certain kind of interactive content encoded in an entirely different way.

The SSM framework, which is also a proposal for MPEG-21 DIA, advances the level of abstraction to develop a flexible methodology for universal adaptation of scalable content, where the adaptation operation is generic enough to be applicable to any type of media having any type of encoding, and does not use any codec-specific code or stylesheets at the adaptation engine. The adaptation engine just needs to be told what the structure of the particular content that goes through it is, and how this content is to be adapted to achieve the desired transcoding operation. This meta-data information is conveyed by the media creator/originator to the engine in XML form. The specific requirements for a recipient are conveyed to the adaptation engine through the outbound constraints specification, also in XML form. With this framework, different adaptation infrastructures are no longer needed for different types of scalable media. For media that is non-standard or for media that do not exist today but would evolve in the future, as long as they conform to the lose bit-stream restrictions that the universal adaptation engine understands, it still becomes possible to adapt it appropriately using SSM adaptation engines.

The main features of the SSM framework are:

- Modeling of scalability in a very generic way that is not too restrictive, yet not so unrestrictive that the number of adaptation choices becomes unacceptably large from the perspective of metadata compactness. (Note that unrestricted adaptation is still supported by *structure driven* adaptation).
- Fully content independent operation, making it easy for new content providers to use. The adaptation infrastructure does not need to change in any way.
- Provides a complete end-to-end delivery solution to multiple recipients minimizing end-to-end redundancy.
- Allows fully secure delivery with encryption since the specifics of the content do not need to be compromised at any point in the delivery chain, either to make adaptation decisions or to do the adpatation.

- Extensible to new types of content with new types of scalability.
- Provides enormous flexibility for adaptation both from the recipient and content creator points of view, for a variety of delivery scenarios.
- Caters to both incremental scalable, exclusive scalable as well as hybrid scalable bit-streams under a common framework.

9. References

[1] *Final Call for Proposals for Digital Item Adaptation*, ISO/IEC JTC1/SC29/WG11/N4683.

[2] *MPEG-21 Digital Item Adaptation AM (v1.0),* ISO/IEC JTC1/SC29/WG11/N4820.

[3] *MPEG-21 Requirements on Digital Item Adaptation*, ISO/IEC JTC1/SC29/WG11/N4684.

[4] David Taubman, "High Performance scalable image compression with EBCOT," *IEEE Transactions on Image Processing*, vol. 9, no. 7, July 2000, pp. 1158-70.

[5] Amir Said and William A. Pearlman, "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, June 1996.

[6] Debargha Mukherjee, "Vector set partitioning and successive refinement VQ for wavelet image and video compression," *PhD thesis*, University of California, Santa Barbara, Aug 1999.

[7] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, no. 12, Dec. 1993.

[8] David S. Taubman and M. W. Marcellin, "JPEG2000: Image Compression Fundamentals, Standards and Practice," *Kluwer Academic Publishers, 2002*.

[9] B. G. Haskell, A. Puri, A. N. Netravali, "Digital Video: An Introduction to MPEG-2," New York: Chapman & Hall, Sept 1996.

[10] Weiping Li, "Overview of Fine Granularity Scalability in MPEG-4 Video Standard," *IEEE Trans. Circuits and Systems for Video Technology*, March 2001, vol. 11, No. 3, pp. 301-317.

[11] *(MPEG-4) Information technology – Coding of audio-visual objects – Part 2: Visual*, ISO/IEC 14496-2-2001.

[12] *(MPEG-4) Information technology – Coding of audio-visual objects – Part 3: Audio, ISO/IEC 14496-3-2001*.

[13] Video Coding for Low Bitrate Communication, ITU-T Recommendation H.263, Nov. 1995.

[14] Video Coding for Low Bitrate Communication, ITU-T SG16/Q.15 H.26L Project, Feb. 2000.

[15] J. Xu, Z. Xiong, S. Li, and Y.-Q. Zhang, "3-D embedded subband coding with optimal truncation (3-D ESCOT)," *J. Applied and Computational Harmonic Analysis: Special Issue on Wavelet Applications in Engineering*, vol. 10, pp. 290-

315, May 2001.

[16] Shih-Ta Hsiang and John W. Woods, "Embedded video coding using motion compensated 3-D subband/wavelet filter bank", Packet Video Workshop, Sardinia, Italy, May 2000.

[17] Shih-Ta Hsiang and John W. Woods, "Embedded video coding using invertible motion compensated 3-D subband/wavelet filter bank," Signal Processing: Image Communications, vol, pp. 705-724, May 2001.

[18] Shih-Ta Hsiang, “Highly Scalable Subband/Wavelet Image and Video Coding,” *Ph.D. Thesis,* Rensselaer Polytechnic Institute, Troy, New York, May 2002.

[19] J. W. Woods and Peisong Chen, “Improved MC-EZBC with quarter-pixel motion vectors,” ISO/IEC JTC1/SC29/WG11, MPEG2002/M8366.

[20] J. W. Woods, Peisong Chen, and Shih-Ta Hsiang, “Exploration experimental results and software,” ISO/IEC JTC1/SC29/WG11, MPEG2002/M8524.

[21] Draft testing procedures for evidence on scalable coding, ISO/IEC JTC1/SC29/WG11, MPEG2002/N4927.

[22] Draft applications and requirements for scalable coding, ISO/IEC JTC1/SC29/WG11, MPEG2002/N4984.

[23] Sylvain Devillers, Myriam Amielh, Thierry Planterose, **“**Bitstream Syntax Description Language (BSDL), Response to the Call for Proposals on MPEG-21 DIA”, ISO/IEC JTC1/SC29/WG11, MPEG2002/M8273.

[24] Sylvain Devillers, “BSDL architecture for multi-step adaptation”, ISO/IEC JTC1/SC29/WG11, MPEG2002/M8523.

[25] Jörg Heuer, Andreas Hutter, Gabriel Panis, Hermann Hellwagner, Harald Kosch, Christian Timmerer (Univ. Klagenfurt), Proposal of a Generic Bitstream Syntax Description Language (g-BSDL), ISO/IEC JTC1/SC29/WG11, MPEG2002/M8291.

[26] Debargha Mukherjee, Amir Said, “Structured Content Independent Scalable Meta-formats (SCISM) for Media Type Agnostic Transcoding: Response to CfP on DIA / MPEG-21,” ISO/IEC JTC1/SC29/WG11, MPEG2002/M8689.

[27] Hermann Hellwagner, Jörg Heuer, Andreas Hutter, Harald Kosch, Gabriel Panis, Christian Timmerer, “Report on Core Experiment on MPEG-21 DIA BSDL focussing on the Generic BSDL (gBSDL),” ISO/IEC JTC1/SC29/WG11, MPEG2002/M8677.

[28] S. J. Wee and J. G. Apostolopoulos, “Secure scalable streaming enabling transcoding without decryption,” *Proc. IEEE Int. Conference on Image Processing*, Thessaloniki, Greece, October 2001, vol. 1, pp. 437-40.

[29] S. J. Wee and J. G. Apostolopoulos, “Secure scalable video streaming for wireless networks,” *Proc. IEEE Int. Conference on Acoustics, Speech and Signal Processing*, Salt Lake City, Utah, May 2001.

[30] D. Wu, Y. T. Hou, W. Zhu, Y.-Q. Zhang, J. M. Peha, “Streaming Media over the Internet: Approaches and Directions,” *IEEE Trans. Circuits and Systems for Video*

Technology, March 2001, vol. 11, No. 3, pp. 282-300.

Appendix A. Complete XML Schemas

This appendix lists the full XML schemas for SSM 2.0.

A.1. Common XML Schema - SSMCommon.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="SSMCommon"
targetNamespace="SSMCommon" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:simpleType name="avarType">
    <xs:restriction base="xs:Name">
      <xs:pattern value="ssm:avar:\c+" />
       </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="compType">
    <xs:restriction base="xs:Name">
      <xs:pattern value="ssm:comp:\c+" />
       </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nonNegativeFloatType">
    <xs:restriction base="xs:float">
          <xs:minInclusive value="0" />
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="nonNegativeIntegerListType">
    <xs:list itemType="xs:nonNegativeInteger" />
  </xs:simpleType>
  <xs:simpleType name="positiveIntegerListType">
    <xs:list itemType="xs:positiveInteger" />
  </xs:simpleType>
  <xs:simpleType name="nonNegativeFloatListType">
    <xs:list itemType="nonNegativeFloatType" />
  </xs:simpleType>
  <xs:simpleType name="floatListType">
    <xs:list itemType="xs:float" />
  </xs:simpleType>
  <xs:simpleType name="addressTypeEnum">
    <xs:restriction base="xs:token">
          <xs:enumeration value="relative" />
      <xs:enumeration value="absolute" />
       </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="endianTypeEnum">
    <xs:restriction base="xs:token">
      <xs:enumeration value="big" />
      <xs:enumeration value="small" />
    </xs:restriction>
  </xs:simpleType>
  <xs:attributeGroup name="attrGroupPosAdd">
      <xs:attribute name="start" type="xs:long" use="required" />
      <xs:attribute name="addressType" type="addressTypeEnum" default="absolute"  />
  </xs:attributeGroup>
  <xs:attributeGroup name="attrGroupPosAddLen">
      <xs:attributeGroup ref="attrGroupPosAdd" />
    <xs:attribute name="length" type="xs:unsignedLong" use="required" />
  </xs:attributeGroup>
  <xs:attributeGroup name="attrGroupPosAddLenBit">
    <xs:attributeGroup ref="attrGroupPosAddLen" />
    <xs:attribute name="bitPos" type="xs:unsignedByte" default="0" />
    <xs:attribute name="signed" type="xs:boolean" default="true" />
    <xs:attribute name="endian" type="endianTypeEnum" default="big" />
  </xs:attributeGroup>
  <xs:simpleType name="operationType">
   <xs:restriction base="xs:token">
    <xs:enumeration value="inverse" />
    <xs:enumeration value="negative" />
    <xs:enumeration value="magnitude" />
    <xs:enumeration value="log" />
    <xs:enumeration value="log10" />
    <xs:enumeration value="exp" />
    <xs:enumeration value="power10" />
    <xs:enumeration value="sqr" />
    <xs:enumeration value="sqrt" />
    <xs:enumeration value="clampZ" />
    <xs:enumeration value="boolIsNZ" />
    <xs:enumeration value="boolIsLEZ" />
    <xs:enumeration value="boolIsGEZ" />
    <xs:enumeration value="boolNOT" />
    <xs:enumeration value="add" />
    <xs:enumeration value="subtract" />
    <xs:enumeration value="absdiff" />
    <xs:enumeration value="multiply" />
    <xs:enumeration value="divide" />
    <xs:enumeration value="maximum" />
    <xs:enumeration value="minimum" />
    <xs:enumeration value="average" />
    <xs:enumeration value="boolOR" />
    <xs:enumeration value="boolAND" />
    <xs:enumeration value="boolXOR" />
    <xs:enumeration value="selector" />
   </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="adapVarType">
   <xs:attribute name="avar" type="avarType" />
     <xs:attribute name="previous" type="xs:boolean" default="false" />
  </xs:complexType>
  <xs:complexType name="compVarType">
     <xs:attribute name="compID" type="compType" />
   <xs:attribute name="indType">
          <xs:simpleType>
             <xs:restriction base="xs:token">
              <xs:enumeration value="inclusionInd" />
              <xs:enumeration value="layerInd" />
              <xs:enumeration value="origLayersInd" />
              <xs:enumeration value="curLayersInd" />
            </xs:restriction>
         </xs:simpleType>
     </xs:attribute>
   <xs:attribute name="param" type="xs:nonNegativeInteger" />
     <xs:attribute name="previous" type="xs:boolean" default="false" />
```

```
</xs:complexType>
<xs:complexType name="stackExpnType">
 <xs:sequence maxOccurs="unbounded">
  <xs:choice>
   <xs:element name="adapVar" type="adapVarType" />
   <xs:element name="compVar" type="compVarType" />
   <xs:element name="constant">
    <xs:complexType>
     <xs:attribute name="value" type="xs:float" />
    </xs:complexType>
   </xs:element>
   <xs:element name="argument">
    <xs:complexType>
     <xs:attribute name="number" type="xs:nonNegativeInteger" />
    </xs:complexType>
   </xs:element>
   <xs:element name="operation">
    <xs:complexType>
     <xs:attribute name="operator" type="operationType" />
    </xs:complexType>
   </xs:element>
  </xs:choice>
 </xs:sequence>
</xs:complexType>
<xs:attributeGroup name="attrGroupLimits">
 <xs:attribute name="lowLimit" type="xs:float" default="1" />
   <xs:attribute name="highLimit" type="xs:float" default="1" />
</xs:attributeGroup>
<xs:complexType name="limitConstraintType">
 <xs:complexContent>
  <xs:extension base="stackExpnType">
       <xs:attributeGroup ref="attrGroupLimits" />
    </xs:extension>
   </xs:complexContent>
</xs:complexType>
 <xs:simpleType name="optimizeType">
    <xs:restriction base="xs:token">
     <xs:enumeration value="maximize" />
     <xs:enumeration value="minimize" />
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="optimizationConstraintType">
    <xs:complexContent>
      <xs:extension base="stackExpnType">
       <xs:attribute name="optimize" type="optimizeType" use="required" />
     </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="adapVarStoreType">
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
       <xs:element name="storedAdapVar" type="adapVarType" />
       <xs:element name="storedCompVar" type="compVarType" />
     </xs:choice>
   </xs:sequence>
</xs:complexType>
```

```
</xs:schema>
```

A.2. Resource Description XML Schema - SSMDescription.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ssm="SSMCommon"
xmlns="SSMDescription" targetNamespace="SSMDescription" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:import namespace="SSMCommon" schemaLocation="SSMCommon.xsd" />
 <xs:simpleType name="exclusiveModelType">
  <xs:restriction base="xs:token">
   <xs:enumeration value="single" />
   <xs:enumeration value="firstAlways" />
   <xs:enumeration value="lastAll" />
   <xs:enumeration value="firstAlwaysLastAll" />
  </xs:restriction>
 </xs:simpleType>
 <xs:complexType name="tierElemInfoType">
  <xs:attribute name="exclusiveFlag" type="xs:boolean" default="false" />
  <xs:attribute name="exclusiveModel" type="exclusiveModelType" default="single" />
  <xs:attribute name="numLayers" type="xs:positiveInteger" use="required" />
  <xs:attribute name="origLayers" type="xs:positiveInteger" use="required" />
  <xs:attribute name="tier" type="xs:nonNegativeInteger" use="required"  />
 </xs:complexType>
 <xs:complexType name="atomTocType">
  <xs:sequence>
   <xs:element name="atomTocEntry" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attributeGroup ref="ssm:attrGroupPosAddLen" />
     <xs:attribute name="indices" type="ssm:indexIntegerListType" use="required" />
    </xs:complexType>
   </xs:element>
   <xs:element name="atomRemovedFromBBox" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:attribute name="indices" type="ssm:indexIntegerListType" use="required" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attributeGroup ref="ssm:attrGroupPosAdd" />
 </xs:complexType>
 <xs:complexType name="compDescriptionDataType">
  <xs:sequence>
   <xs:element name="tierInfo">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="tierElemInfo" type="tierElemInfoType" maxOccurs="unbounded" />
     </xs:sequence>
     <xs:attribute name="numTiers" type="xs:positiveInteger" />
    </xs:complexType>
   </xs:element>
   <xs:element name="atomToc" type="atomTocType" />
  </xs:sequence>
  <xs:attribute name="compID" type="ssm:compType" use="required" />
 </xs:complexType>
```

```
<xs:complexType name="marginalDistType">
 <xs:simpleContent>
  <xs:extension base="ssm:nonNegativeFloatListType">
   <xs:attribute name="dims" type="xs:positiveInteger" />
   <xs:attribute name="dimToTierMap" type="ssm:nonNegativeIntegerListType" use="required" />
   <xs:attribute name="distType">
    <xs:simpleType>
     <xs:restriction base="xs:token">
      <xs:enumeration value="nonDecreasing" />
      <xs:enumeration value="nonIncreasing" />
      <xs:enumeration value="nonMonotonic" />
      <xs:enumeration value="constant" />
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="featureDistType">
 <xs:sequence>
  <xs:element name="marginalDist" type="marginalDistType" maxOccurs="unbounded" />
 </xs:sequence>
</xs:complexType>
<xs:complexType name="featureDataType">
 <xs:sequence>
  <xs:element name="components">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="component" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="compID" type="ssm:compType" />
       <xs:attribute name="numLayers" type="ssm:positiveIntegerListType" use="optional" />
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="refFeatureValue" type="ssm:nonNegativeFloatType" />
  <xs:element name="emptyFeatureDist" type="ssm:nonNegativeFloatType" />
  <xs:element name="featureDist" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:complexContent>
     <xs:extension base="featureDistType">
      <xs:attribute name="emptyComponentTiers" type="ssm:nonNegativeIntegerListType"
default="" />
     </xs:extension>
    </xs:complexContent>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="avar" type="ssm:avarType" use="required" />
</xs:complexType>
<xs:complexType name="combAvarType">
 <xs:complexContent>
  <xs:extension base="ssm:stackExpnType">
   <xs:attribute name="avar" type="ssm:avarType" use="required" />
```

```
     <xs:attribute name="numArguments" type="xs:nonNegativeInteger" default="0" />
    </xs:extension>
  </xs:complexContent>
  </xs:complexType>
   <xs:complexType name="LUTAvarType">
   <xs:sequence>
     <xs:element name="axisValues" maxOccurs="unbounded">
      <xs:complexType>
        <xs:attribute name="grid" type="ssm:floatListType" use="required" />
        <xs:attribute name="axis" type="xs:nonNegativeInteger" use="required" />
      </xs:complexType>
     </xs:element>
     <xs:element name="content">
      <xs:complexType>
        <xs:sequence>
         <xs:element name="marginalDist" type="marginalDistType" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
     </xs:element>
   </xs:sequence>
   <xs:attribute name="numAxes" type="xs:positiveInteger" default="1" />
   <xs:attribute name="avar" type="ssm:avarType" use="required" />
   <xs:attribute name="interpolationMethod" default="linear">
   <xs:simpleType>
     <xs:restriction base="xs:token">
      <xs:enumeration value="linear" />
      <xs:enumeration value="round" />
      <xs:enumeration value="ceil" />
      <xs:enumeration value="floor" />
     </xs:restriction>
   </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="resourceEditType">
  <xs:complexContent>
   <xs:extension base="ssm:stackExpnType">
     <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
     <xs:attribute name="outLength" use="optional">
      <xs:simpleType>
        <xs:union memberTypes="ssm:avarType xs:nonNegativeInteger" />
      </xs:simpleType>
     </xs:attribute>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>
<xs:simpleType name="invalidPointerHandlingType">
  <xs:restriction base="xs:token">
   <xs:enumeration value="moveUp" />
   <xs:enumeration value="moveDown" />
   <xs:enumeration value="zeroOut" />
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="offsetReferenceDataType">
  <xs:sequence>
   <xs:element name="offsetEntry" maxOccurs="unbounded">
     <xs:complexType>
```

```
          <xs:attribute name="value" type="xs:long" use="required" />
          <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
          <xs:attribute name="invalidPointerHandling" type="invalidPointerHandlingType"
default="moveUp" />
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attributeGroup ref="ssm:attrGroupPosAdd" />
    <xs:attribute name="invalidPointerHandling" type="invalidPointerHandlingType"
default="moveUp" />
  </xs:complexType>
  <xs:complexType name="sequenceDataType">
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="writeField" minOccurs="0">
          <xs:complexType>
            <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
            <xs:attribute name="type" default="count">
              <xs:simpleType>
                <xs:restriction base="xs:token">
                  <xs:enumeration value="count" />
                  <xs:enumeration value="seqValue" />
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element name="countField">
          <xs:complexType>
            <xs:attributeGroup ref="ssm:attrGroupPosAddLenBit" />
            <xs:attribute name="write" type="xs:boolean" default="true" />
          </xs:complexType>
          </xs:element>
        <xs:element name="subSequence" type="sequenceDataType" />
        <xs:element name="countOnly">
          <xs:complexType />
        </xs:element>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="startValue" type="xs:long" use="optional" default="0" />
    <xs:attribute name="stepValue" type="xs:long" use="optional" default="1" />
    <xs:attribute name="modulo" type="xs:long" use="optional"/>
    <xs:attribute name="pack" type="xs:boolean" use="optional" default="true" />
    <xs:attribute name="relativeStart" type="xs:boolean" use="optional" default="true" />
  </xs:complexType>
  <xs:element name="SSMDescription">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="parcelData">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="parcel" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="componentData">
                      <xs:complexType>
            <xs:sequence>
             <xs:element name="component" type="compDescriptionDataType"
maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
           <xs:unique name="compID">
            <xs:selector xpath="component" />
            <xs:field xpath="@compID" />
           </xs:unique>
          </xs:element>
          <xs:element name="featureData" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="feature" type="featureDataType" minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="combAvarData" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="combAvar" type="combAvarType" minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="LUTAvarData" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
              <xs:element name="LUTAvar" type="LUTAvarType" minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="creatorLimitConstraints" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="limitConstraint" type="ssm:limitConstraintType" minOccurs="0"
maxOccurs="unbounded" />
             </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="adapVarStore" type="ssm:adapVarStoreType" minOccurs="0"
maxOccurs="1" />
          <xs:element name="resourceEditData" minOccurs="0">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="resourceEdit" type="resourceEditType" minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="parcelID" type="xs:unsignedLong" use="optional" />
        </xs:complexType>
        <xs:unique name="avar">
```

```
         <xs:selector
xpath="componentData/component/featureData/feature|combAvarData/combAvar|featureData/featu
re|LUTAvarData/LUTAvar" />
         <xs:field xpath="@avar" />
        </xs:unique>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
     <xs:unique name="parcelID">
      <xs:selector xpath="parcel" />
      <xs:field xpath="@parcelID" />
     </xs:unique>
    </xs:element>
    <xs:element name="globalFeatureData" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="globalFeature" type="featureDataType" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="globalCombAvarData" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="globalCombAvar" type="combAvarType" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="globalLUTAvarData" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="globalLUTAvar" type="LUTAvarType" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="globalCreatorLimitConstraints" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="globalLimitConstraint" type="ssm:limitConstraintType" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="codecOffsetData" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="offsetReference" type="offsetReferenceDataType" minOccurs="0"
maxOccurs="unbounded" />
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="sequenceData" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
```

```
          <xs:element name="sequence" type="sequenceDataType" minOccurs="0"
maxOccurs="unbounded" />
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
</xs:schema>
```

A.3. Outbound Constraints XML Schema - SSMAdaptReq.xsd

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ssm="SSMCommon"
xmlns="SSMAdaptReq" targetNamespace="SSMAdaptReq" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="SSMCommon" schemaLocation="SSMCommon.xsd" />
  <xs:simpleType name="profileType">
    <xs:restriction base="xs:Name">
      <xs:pattern value="ssm:prof:\c+" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="adapVarDrivenAdaptType">
    <xs:sequence>
      <xs:element name="limitConstraint" type="ssm:limitConstraintType" minOccurs="0"
maxOccurs="unbounded" />
      <xs:element name="optimizationConstraint" type="ssm:optimizationConstraintType"
minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="structureDrivenAdaptType">
    <xs:sequence>
      <xs:element name="adaptationPoint" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="layers" type="ssm:nonNegativeIntegerListType" use="required" />
          <xs:attribute name="compID" type="ssm:compType" use="required" />
          <xs:attribute name="incType" default="bboxInc">
            <xs:simpleType>
              <xs:restriction base="xs:token">
                <xs:enumeration value="bboxDrop" />
                <xs:enumeration value="bboxInc" />
                <xs:enumeration value="atomInc" />
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="SSMAdaptReq">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="profile" maxOccurs="unbounded">
          <xs:complexType>
          <xs:sequence>
           <xs:element name="parcel" maxOccurs="unbounded">
            <xs:complexType>
             <xs:sequence>
              <xs:choice>
               <xs:element name="adapVarDriven" type="adapVarDrivenAdaptType" />
               <xs:element name="structureDriven" type="structureDrivenAdaptType" />
              </xs:choice>
              <xs:element name="adapVarStore" type="ssm:adapVarStoreType" minOccurs="0"
maxOccurs="1" />
             </xs:sequence>
             <xs:attribute name="parcelID" type="xs:unsignedLong" use="optional" />
             <xs:attribute name="requestType" default="SSM">
              <xs:simpleType>
               <xs:restriction base="xs:token">
                <xs:enumeration value="SSM" />
                <xs:enumeration value="unstructured" />
                <xs:enumeration value="stopProfile" />
                <xs:enumeration value="asIs" />
               </xs:restriction>
              </xs:simpleType>
             </xs:attribute>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         <xs:attribute name="profileID" type="profileType" />
        </xs:complexType>
        <xs:unique name="parcelID">
         <xs:selector xpath="parcel" />
         <xs:field xpath="@parcelID" />
        </xs:unique>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="adaptType" default="terminal">
       <xs:simpleType>
        <xs:restriction base="xs:token">
         <xs:enumeration value="terminal" />
         <xs:enumeration value="midstream" />
        </xs:restriction>
       </xs:simpleType>
      </xs:attribute>
     </xs:complexType>
     <xs:unique name="profileID">
      <xs:selector xpath="profile" />
      <xs:field xpath="@profileID" />
     </xs:unique>
    </xs:element>
</xs:schema>
```

Appendix B. Sample XML files

In this appendix, we present two sample XML files, one for the resource description, and the other for the outbound constraints, along with an adapted resource description XML generated by applying adaptation based on the outbound constraints XML to the original resource description XML, using an early version of the SSM adaptation engine software that we are developing.

B.1. Resource Description XML - SSMDescription_ex.xml

The resource description XML below describes a bit-stream with two parcels. Each parcel has two components *ssm:comp:myImage* and *ssm:comp:myAudio*. *ssm:comp:myImage* is a two-tier scalable component with 4x5 layers in both parcels, while *ssm:comp:myAudio* is a single tier scalable component with 6 layers in the first parcel, and 5 layers in the second. Elemental features defined for the image component are *ssm:avar:imageCodesize*, *ssm:avar:imageDistortion*, and *ssm:avar:imageResolution*, while those defined for the audio component are *ssm:avar:audioCodesize* and *ssm:avar:audioDistortion*. In addition, a product feature *ssm:avar:perceptualRichness* is defined jointly over the image and audio components. Based on the elemental features, the global combination variable *ssm:avar:codesize* is defined as the sum of *ssm:avar:imageCodesize* and *ssm:avar:audioCodesize*. Another global combination variable *ssm:avar:lagrangian* that takes two arguments, is defined as: $(argument_0 \times ssm:avar:imageDistortion+ssm:avar:audioDistortion)+argument_1 \times ssm:avar:codesize$. This is essentially a $D+\lambda.R$ type metric where D is a weighted distortion measure of the image and audio components, with the weight and the lagrangian parameter being the two arguments. There is also a global LUT variable *ssm:avar:lagrangianLUT*, that yields an appropriate lagrangian parameter λ from two inputs related to total desired transmission time and bandwidth. A fourth global combination variable *ssm:avar:myImageImpliesAudio*, which is Boolean, is defined based on component inclusion indicator variables for the two components to indicate whether audio component inclusion implies that the image component is also included. A global creator enforced limit constraint *ssm:avar:myImageImpliesAudio* = 1 is defined based on this combination variable to enforce that if the image component is included the audio component must be included too.

```
<?xml version="1.0" ?>
<SSMDescription xsi:schemaLocation="SSMDescription SSMDescription.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="SSMDescription"
xmlns:ssm="SSMCommon">
 <parcelData>
  <parcel parcelID="0">
   <componentData>
    <component compID="ssm:comp:myImage">
     <tierInfo numTiers="2">
      <tierElemInfo numLayers="4" origLayers="4" exclusiveFlag="false" tier="0" />
      <tierElemInfo numLayers="5" origLayers="5" exclusiveFlag="false" tier="1" />
     </tierInfo>
     <atomToc start="0" addressType="absolute">
      <atomTocEntry indices="0 0" start="15000" addressType="relative" length="512" />
      <atomTocEntry indices="0 1" start="15600" addressType="relative" length="360" />
      <atomTocEntry indices="0 2" start="16347" addressType="relative" length="2000" />
      <atomTocEntry indices="0 3" start="18400" addressType="relative" length="2536" />
      <atomTocEntry indices="0 4" start="21506" addressType="relative" length="3078" />
      <atomTocEntry indices="1 0" start="25000" addressType="relative" length="1006" />
      <atomTocEntry indices="1 1" start="26120" addressType="relative" length="1878" />
      <atomTocEntry indices="1 2" start="27303" addressType="relative" length="2663" />
      <atomTocEntry indices="1 3" start="30000" addressType="relative" length="3549" />
      <atomTocEntry indices="1 4" start="36000" addressType="relative" length="4812" />
      <atomTocEntry indices="2 0" start="40904" addressType="relative" length="1470" />
      <atomTocEntry indices="2 1" start="42655" addressType="relative" length="2351" />
      <atomTocEntry indices="2 2" start="45101" addressType="relative" length="3534" />
      <atomTocEntry indices="2 3" start="48709" addressType="relative" length="4915" />
      <atomTocEntry indices="2 4" start="53810" addressType="relative" length="6002" />
      <atomTocEntry indices="3 0" start="60000" addressType="relative" length="2029" />
      <atomTocEntry indices="3 1" start="62029" addressType="relative" length="3147" />
      <atomTocEntry indices="3 2" start="65200" addressType="relative" length="4258" />
      <atomTocEntry indices="3 3" start="69458" addressType="relative" length="5307" />
      <atomTocEntry indices="3 4" start="74819" addressType="relative" length="7003" />
     </atomToc>
    </component>
    <component compID="ssm:comp:myAudio">
     <tierInfo numTiers="1">
      <tierElemInfo numLayers="6" origLayers="6" exclusiveFlag="false" tier="0" />
     </tierInfo>
     <atomToc start="90000" addressType="relative">
      <atomTocEntry indices="0" start="1178" addressType="relative" length="789" />
      <atomTocEntry indices="1" start="1967" addressType="relative" length="1745" />
      <atomTocEntry indices="2" start="3825" addressType="relative" length="2840" />
      <atomTocEntry indices="3" start="6778" addressType="relative" length="4173" />
      <atomTocEntry indices="4" start="10951" addressType="relative" length="5281" />
      <atomTocEntry indices="5" start="16378" addressType="relative" length="6989" />
     </atomToc>
    </component>
   </componentData>
   <featureData>
    <feature avar="ssm:avar:imageCodesize">
     <components>
      <component compID="ssm:comp:myImage" />
     </components>
     <refFeatureValue>1.0</refFeatureValue>
     <emptyFeatureDist>0.0</emptyFeatureDist>
     <featureDist emptyComponentTiers="">
      <marginalDist dims="2" dimToTierMap="0 1" distType="nonDecreasing">180 260 390 470 680
310 390 480 590 715 405 539 676 799 883 478 626 797 899 1067</marginalDist>
     </featureDist>
    </feature>
    <feature avar="ssm:avar:imageDistortion">
     <components>
      <component compID="ssm:comp:myImage" />
     </components>
     <refFeatureValue>1.0</refFeatureValue>
     <emptyFeatureDist>2000.0</emptyFeatureDist>
     <featureDist emptyComponentTiers="">
      <marginalDist dims="2" dimToTierMap="0 1" distType="nonIncreasing">800 600 513 321 198
750 553 420 314 154 679 521 390 247 113 537 472 355 206 84</marginalDist>
     </featureDist>
    </feature>
    <feature avar="ssm:avar:audioCodesize">
     <components>
      <component compID="ssm:comp:myAudio" />
     </components>
     <refFeatureValue>1.0</refFeatureValue>
```

```
        <emptyFeatureDist>0</emptyFeatureDist>
        <featureDist emptyComponentTiers="">
          <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">155 231 367 449 673
757</marginalDist>
        </featureDist>
      </feature>
      <feature avar="ssm:avar:audioDistortion">
        <components>
          <component compID="ssm:comp:myAudio" />
        </components>
        <refFeatureValue>1.0</refFeatureValue>
        <emptyFeatureDist>600</emptyFeatureDist>
        <featureDist emptyComponentTiers="">
          <marginalDist dims="1" dimToTierMap="0" distType="nonIncreasing">400 356 267 100 50
22</marginalDist>
        </featureDist>
      </feature>
      <feature avar="ssm:avar:perceptualRichness">
        <components>
          <component compID="ssm:comp:myImage" />
          <component compID="ssm:comp:myAudio" />
        </components>
        <refFeatureValue>10.0</refFeatureValue>
        <emptyFeatureDist>0.0</emptyFeatureDist>
        <featureDist emptyComponentTiers="0 1">
          <marginalDist dims="1" dimToTierMap="2">1 2 3 5 8 10</marginalDist>
        </featureDist>
        <featureDist emptyComponentTiers="2">
          <marginalDist dims="1" dimToTierMap="0">1 1.2 1.5 2</marginalDist>
          <marginalDist dims="1" dimToTierMap="1">1 2 3 4 4.5</marginalDist>
        </featureDist>
        <featureDist emptyComponentTiers="">
          <marginalDist dims="3" dimToTierMap="0 1 2">1 2 3 4 4 5 2 3 3 4 5 5 2 3 3 4 5 6 3 3 4 4 5 6 3 3
4 5 6 6 3 4 5 6 6 7 4 5 5 6 7 7 4 5 5 6 7 8 5 5 6 6 7 8 5 5 6 7 8 8 6 7 8 9 9 10 7 8 8 9 10 10 7 8 8 9 10 11 8 8
9 9 10 11 8 8 9 10 11 11 11 12 13 14 14 15 12 13 13 14 15 15 12 13 13 14 15 16 13 13 14 14 15 16 13 13
14 15 16 16</marginalDist>
        </featureDist>
      </feature>
    </featureData>
    <combAvarData />
    <resourceEditData>
      <resourceEdit start="10000" addressType="absolute" length="8">
        <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="0" />
      </resourceEdit>
      <resourceEdit start="2" addressType="relative" length="8">
        <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="1" />
      </resourceEdit>
      <resourceEdit start="4" addressType="relative" length="8">
        <ssm:compVar compID="ssm:comp:myAudio" indType="layerInd" param="0" />
      </resourceEdit>
    </resourceEditData>
  </parcel>
  <parcel parcelID="1">
    <componentData>
      <component compID="ssm:comp:myImage">
        <tierInfo numTiers="2">
          <tierElemInfo numLayers="4" origLayers="4" exclusiveFlag="false" tier="0" />
          <tierElemInfo numLayers="5" origLayers="5" exclusiveFlag="false" tier="1" />
        </tierInfo>
        <atomToc start="30000" addressType="relative">
          <atomTocEntry indices="0 0" start="1000" addressType="relative" length="512" />
          <atomTocEntry indices="0 1" start="1600" addressType="relative" length="360" />
          <atomTocEntry indices="0 2" start="2347" addressType="relative" length="2000" />
          <atomTocEntry indices="0 3" start="4400" addressType="relative" length="2536" />
          <atomTocEntry indices="0 4" start="7506" addressType="relative" length="3078" />
          <atomTocEntry indices="1 0" start="11000" addressType="relative" length="1006" />
          <atomTocEntry indices="1 1" start="12120" addressType="relative" length="1878" />
          <atomTocEntry indices="1 2" start="13303" addressType="relative" length="2663" />
          <atomTocEntry indices="1 3" start="16000" addressType="relative" length="3549" />
          <atomTocEntry indices="1 4" start="22000" addressType="relative" length="4812" />
          <atomTocEntry indices="2 0" start="26904" addressType="relative" length="1470" />
          <atomTocEntry indices="2 1" start="28655" addressType="relative" length="2351" />
          <atomTocEntry indices="2 2" start="31101" addressType="relative" length="3534" />
          <atomTocEntry indices="2 3" start="34709" addressType="relative" length="4915" />
          <atomTocEntry indices="2 4" start="39810" addressType="relative" length="6002" />
          <atomTocEntry indices="3 0" start="46000" addressType="relative" length="2029" />
          <atomTocEntry indices="3 1" start="48029" addressType="relative" length="3147" />
          <atomTocEntry indices="3 2" start="51200" addressType="relative" length="4258" />
          <atomTocEntry indices="3 3" start="55458" addressType="relative" length="5307" />
          <atomTocEntry indices="3 4" start="60819" addressType="relative" length="7003" />
        </atomToc>
      </component>
      <component compID="ssm:comp:myAudio">
        <tierInfo numTiers="1">
          <tierElemInfo numLayers="5" origLayers="5" exclusiveFlag="false" tier="0" />
        </tierInfo>
        <atomToc start="70000" addressType="relative">
          <atomTocEntry indices="0" start="1178" addressType="relative" length="789" />
          <atomTocEntry indices="1" start="1967" addressType="relative" length="1745" />
          <atomTocEntry indices="2" start="3825" addressType="relative" length="2840" />
          <atomTocEntry indices="3" start="6778" addressType="relative" length="4173" />
          <atomTocEntry indices="4" start="10951" addressType="relative" length="5281" />
        </atomToc>
      </component>
    </componentData>
    <featureData>
      <feature avar="ssm:avar:imageResolution">
        <components>
          <component compID="ssm:comp:myImage" />
        </components>
        <refFeatureValue>1.0</refFeatureValue>
        <emptyFeatureDist>0.0</emptyFeatureDist>
        <featureDist emptyComponentTiers="">
          <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">100 200 400
800</marginalDist>
          <marginalDist dims="1" dimToTierMap="1" distType="constant">1.0</marginalDist>
        </featureDist>
      </feature>
      <feature avar="ssm:avar:imageCodesize">
        <components>
          <component compID="ssm:comp:myImage" />
        </components>
```

```
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dims="2" dimToTierMap="0 1" distType="nonDecreasing">120 240 350 400 580
240 300 420 510 625 315 456 600 720 800 378 526 720 823 999</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:imageDistortion">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>2000.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dims="2" dimToTierMap="0 1" distType="nonIncreasing">1400 1000 813 621
428 1200 803 647 531 272 1000 671 436 317 183 699 475 355 206 137</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioCodesize">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">105 211 323 456
641</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioDistortion">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>600</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dims="1" dimToTierMap="0" distType="nonIncreasing">400 356 237 160
90</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:perceptualRichness">
          <components>
            <component compID="ssm:comp:myImage" />
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>10.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="0 1">
            <marginalDist dims="1" dimToTierMap="2">2 3 5 8 10</marginalDist>
          </featureDist>
          <featureDist emptyComponentTiers="2">
            <marginalDist dims="1" dimToTierMap="0">1 1.2 1.5 2</marginalDist>
            <marginalDist dims="1" dimToTierMap="1">1 2 3 4 4.5</marginalDist>
          </featureDist>
          <featureDist emptyComponentTiers="">
        <marginalDist dims="3" dimToTierMap="0 1 2">2 3 4 4 5 3 3 4 5 5 3 3 4 5 6 3 4 4 5 6 3 4 5 6 6 4
5 6 6 7 5 5 6 7 7 5 5 6 7 8 5 6 6 7 8 5 6 7 8 8 7 8 9 9 10 8 8 9 10 10 8 8 9 10 11 8 9 9 10 11 8 9 10 11 11 12
13 14 14 15 13 13 14 15 15 13 13 14 15 16 13 14 14 15 16 13 14 15 16 16</marginalDist>
      </featureDist>
     </feature>
    </featureData>
    <combAvarData />
    <resourceEditData>
     <resourceEdit start="120050" addressType="absolute" length="8">
      <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="0" />
     </resourceEdit>
     <resourceEdit start="2" addressType="relative" length="8">
      <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="1" />
     </resourceEdit>
     <resourceEdit start="10" addressType="relative" length="8">
      <ssm:compVar compID="ssm:comp:myAudio" indType="layerInd" param="0" />
     </resourceEdit>
    </resourceEditData>
   </parcel>
  </parcelData>
  <globalFeatureData>
   <globalFeature avar="ssm:avar:imageResolution">
    <components>
     <component compID="ssm:comp:myImage" numLayers="5 5" />
    </components>
    <refFeatureValue>1.0</refFeatureValue>
    <emptyFeatureDist>0.0</emptyFeatureDist>
    <featureDist emptyComponentTiers="">
     <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">100 200 400 800
1600</marginalDist>
     <marginalDist dims="1" dimToTierMap="1" distType="constant">1.0</marginalDist>
    </featureDist>
   </globalFeature>
  </globalFeatureData>
  <globalCombAvarData>
   <globalCombAvar avar="ssm:avar:codesize">
    <ssm:adapVar avar="ssm:avar:audioCodesize" />
    <ssm:adapVar avar="ssm:avar:imageCodesize" />
    <ssm:operation operator="add" />
   </globalCombAvar>
   <globalCombAvar avar="ssm:avar:lagrangian" numArguments="2">
    <ssm:adapVar avar="ssm:avar:audioDistortion" />
    <ssm:adapVar avar="ssm:avar:imageDistortion" />
    <ssm:argument number="0" />
    <ssm:operation operator="multiply" />
    <ssm:operation operator="add" />
    <ssm:adapVar avar="ssm:avar:codesize" />
    <ssm:argument number="1" />
    <ssm:operation operator="multiply" />
    <ssm:operation operator="add" />
   </globalCombAvar>
   <globalCombAvar avar="ssm:avar:myImageImpliesAudio">
    <ssm:compVar compID="ssm:comp:myAudio" indType="inclusionInd" />
    <ssm:compVar compID="ssm:comp:myImage" indType="inclusionInd" />
    <ssm:operation operator="boolNOT" />
    <ssm:operation operator="boolOR" />
    </globalCombAvar>
  </globalCombAvarData>
  <globalLUTAvarData>
    <globalLUTAvar avar="ssm:avar:lagrangianLUT" numAxes="2">
      <axisValues axis="0" grid="25 50 100" />
      <axisValues axis="1" grid="10 30 100 300" />
      <content>
        <marginalDist dims="2" dimToTierMap="0 1">0.9 0.8 0.7 0.6 0.5 0.4 0.8 0.7 0.6 0.5 0.4 0.3 0.7 0.6
0.5 0.4 0.3 0.2 0.6 0.5 0.4 0.3 0.2 0.1 0.5 0.4 0.3 0.2 0.1 0.1</marginalDist>
      </content>
    </globalLUTAvar>
  </globalLUTAvarData>
  <globalCreatorLimitConstraints>
    <globalLimitConstraint lowLimit="1" highLimit="1">
      <ssm:adapVar avar="ssm:avar:myImageImpliesAudio" />
    </globalLimitConstraint>
  </globalCreatorLimitConstraints>
  <codecOffsetData>
    <offsetReference start="85000" addressType="relative">
      <offsetEntry value="10000" start="2" length="16" addressType="relative"
invalidPointerHandling="moveUp" />
      <offsetEntry value="20000" start="4" length="16" addressType="relative"
invalidPointerHandling="moveUp" />
    </offsetReference>
    <offsetReference start="3000" addressType="relative">
      <offsetEntry value="-8000" start="2" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
      <offsetEntry value="30000" start="4" bitPos="4" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
      <offsetEntry value="50000" start="7" length="20" addressType="relative"
invalidPointerHandling="moveUp" />
    </offsetReference>
  </codecOffsetData>
  <sequenceData>
    <sequence startValue="10" stepValue="2">
      <countField start="10000" length="16" addressType="absolute" />
      <countField start="5000" length="16" addressType="relative" />
      <countField start="1000" length="16" addressType="relative" />
      <countField start="2000" length="16" addressType="relative" />
    </sequence>
    <sequence startValue="0" stepValue="1">
      <countField start="20000" length="8" addressType="absolute" write="false" />
      <countField start="6000" length="8" addressType="relative" write="false" />
      <countField start="4000" length="8" addressType="relative" write="false" />
      <countField start="3000" length="8" addressType="relative" write="false" />
      <writeField start="2000" length="8" addressType="relative" />
      <countField start="4000" length="8" addressType="relative" write="false" />
      <countField start="8000" length="8" addressType="relative" write="false" />
    </sequence>
    <sequence startValue="0" stepValue="6">
      <writeField start="60000" length="8" addressType="absolute" type="seqValue" />
      <subSequence startValue="0" stepValue="1" pack="false">
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
      </subSequence>
      <subSequence startValue="0" stepValue="1" pack="false" relativeStart="false">
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countOnly />
        <countOnly />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countOnly />
        <countOnly />
      </subSequence>
      <subSequence startValue="0" stepValue="1" pack="false" modulo="8">
        <countField start="60000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countField start="1000" length="8" addressType="relative" write="true" />
        <countOnly />
        <countOnly />
        <countOnly />
        <writeField start="4000" length="8" addressType="relative" type="count" />
      </subSequence>
    </sequence>
  </sequenceData>
</SSMDescription>
```

B.2. Outbound Constraints XML - SSMAdaptReq_ex.xml

The outbound constraints XML that operates on the above resource description file includes two profiles to be packaged as a single bit-stream using terminal adaptation type. The first profile uses adaptation variable based adaptation requests, while the second uses structure-driven adaptation. The first parcel of the first profile uses a limit constraint that the *ssm:avar:imageResolution* feature must be in the range 400-960. The optimization constraint attempts to maximize *ssm:avar:perceptualRichness* + β. *ssm:avar:codesize* for a given $\beta = -0.1$. For the second and all subsequent parcels, the limit constraint based on *ssm:avar:imageResolution* changes to that the feature *ssm:avar:imageResolution* must be exactly equal to the *ssm:avar:imageResolution* variable for the previous parcel. Further, the optimization constraint attempts to minimize *ssm:avar:lagrangian* with two parameters, the weight for the weighted distortion measure (1.5), and the lagrangian parameter. The latter in turn is obtained from a LUT in the description XML that takes two inputs: total desired transmission time (25) and bandwidth (10). Thus, from the second parcel onwards, the image resolution of the adapted version remains the same, while a weighted rate-distortion optimization is performed. For the second profile, all parcels use the structure driven adaptation, where the adaptation point is specified explicitly.

```
<?xml version="1.0" ?>
<SSMAdaptReq adaptType="terminal" xsi:schemaLocation="SSMAdaptReq SSMAdaptReq.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="SSMAdaptReq"
xmlns:ssm="SSMCommon">
 <profile profileID="ssm:prof:terminal-1">
  <parcel parcelID="0">
   <adapVarDriven>
```

```
        <limitConstraint lowLimit="400" highLimit="960">
          <ssm:adapVar avar="ssm:avar:imageResolution" />
        </limitConstraint>
        <optimizationConstraint optimize="maximize">
          <ssm:constant value="-0.1" />
          <ssm:adapVar avar="ssm:avar:codesize" />
          <ssm:operation operator="multiply" />
          <ssm:adapVar avar="ssm:avar:perceptualRichness" />
          <ssm:boperation operator="add" />
        </optimizationConstraint>
      </adapVarDriven>
      <adapVarStore>
        <ssm:storedAdapVar avar="ssm:avar:imageResolution" />
      </adapVarStore>
     </parcel>
    <parcel parcelID="1">
      <adapVarDriven>
        <limitConstraint lowLimit="0" highLimit="0">
          <ssm:adapVar avar="ssm:avar:imageResolution" />
          <ssm:adapVar avar="ssm:avar:imageResolution" previous="true" />
          <ssm:operation operator="subtract" />
        </limitConstraint>
        <optimizationConstraint optimize="minimize">
          <ssm:constant value="10" />
          <ssm:constant value="25" />
          <ssm:adapVar avar="ssm:avar:lagrangianLUT" />
          <ssm:constant value="1.50" />
          <ssm:adapVar avar="ssm:avar:lagrangian" />
      </optimizationConstraint>
      </adapVarDriven>
      <adapVarStore>
        <ssm:storedAdapVar avar="ssm:avar:imageResolution" />
      </adapVarStore>
    </parcel>
  </profile>
  <profile profileID="ssm:prof:terminal-2">
    <parcel parcelID="0">
      <structureDriven>
        <adaptationPoint compID="ssm:comp:myImage" layers="2 2" incType="bboxDrop" />
        <adaptationPoint compID="ssm:comp:myAudio" layers="5" incType="bboxInc" />
     </structureDriven>
    </parcel>
    <parcel parcelID="1">
      <structureDriven>
        <adaptationPoint compID="ssm:comp:myImage" layers="2 3" incType="bboxDrop" />
        <adaptationPoint compID="ssm:comp:myAudio" layers="4" incType="bboxInc" />
      </structureDriven>
    </parcel>
  </profile>
</SSMAdaptReq>
```

B.3. Adapted Resource Description XML – SSMDescription_ex2.xml

When the SSM adaptation engine software is applied to the resource description XML and the outbound constraints XML, the decisions are made, and the following adapted resource description is generated. For the first parcel, the two profiles generate adaptation points (4,1) and (2,3) for the image component. While the bounding box is (4,3), because the adaptation type is *terminal* the atoms with indices (2,1), (2,2), (3,1) and (3,2) are actually removed from the bounding box, and is shown in the <atomRemovedFromBBox> elements of <atomToc>. For the audio component, the two profiles yield 2 and 5 layers respectively, and so in the packaged bit-stream all 5 layers are kept. For the second parcel, the adaptation points for the two profiles for the image component are (4,4) and (2,2) respectively. The bounding box transmitted is (4,4). For the audio component, the two profiles yield 3 and 4 layers respectively, and so 4 layers are kept in the adapted bit-stream.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SSMDescription xmlns="SSMDescription" xmlns:ssm="SSMCommon"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="SSMDescription
SSMDescription.xsd">
  <parcelData>
    <parcel parcelID="0">
      <componentData>
        <component compID="ssm:comp:myImage">
          <tierInfo numTiers="2">
            <tierElemInfo exclusiveFlag="false" numLayers="4" origLayers="4" tier="0" />
            <tierElemInfo exclusiveFlag="false" numLayers="3" origLayers="5" tier="1" />
          </tierInfo>
          <atomToc addressType="absolute" start="0">
            <atomTocEntry addressType="relative" indices="0 0" length="512" start="15000" />
            <atomTocEntry addressType="relative" indices="0 1" length="360" start="15600" />
            <atomTocEntry addressType="relative" indices="0 2" length="2000" start="16347" />
            <atomTocEntry addressType="relative" indices="1 0" length="1006" start="19386" />
            <atomTocEntry addressType="relative" indices="1 1" length="1878" start="20506" />
            <atomTocEntry addressType="relative" indices="1 2" length="2663" start="21689" />
            <atomTocEntry addressType="relative" indices="2 0" length="1470" start="26929" />
            <atomTocEntry addressType="relative" indices="3 0" length="2029" start="29223" />
            <atomRemovedFromBBox indices="2 1" />
            <atomRemovedFromBBox indices="2 2" />
            <atomRemovedFromBBox indices="3 1" />
            <atomRemovedFromBBox indices="3 2" />
          </atomToc>
        </component>
        <component compID="ssm:comp:myAudio">
          <tierInfo numTiers="1">
            <tierElemInfo exclusiveFlag="false" numLayers="5" origLayers="6" tier="0" />
          </tierInfo>
          <atomToc addressType="relative" start="39508">
            <atomTocEntry addressType="relative" indices="0" length="789" start="1178" />
            <atomTocEntry addressType="relative" indices="1" length="1745" start="1967" />
            <atomTocEntry addressType="relative" indices="2" length="2840" start="3825" />
            <atomTocEntry addressType="relative" indices="3" length="4173" start="6778" />
            <atomTocEntry addressType="relative" indices="4" length="5281" start="10951" />
          </atomToc>
        </component>
      </componentData>
      <featureData>
        <feature avar="ssm:avar:imageCodesize">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0 1" dims="2" distType="nonDecreasing">180 260 390 310 390
480 405 539 676 478 626 797</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:imageDistortion">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>2000.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0 1" dims="2" distType="nonIncreasing">800 600 513 750 553
420 679 521 390 537 472 355</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioCodesize">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0" dims="1" distType="nonDecreasing">155 231 367 449
673</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioDistortion">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>600</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0" dims="1" distType="nonIncreasing">400 356 267 100
50</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:perceptualRichness">
          <components>
            <component compID="ssm:comp:myImage" />
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>10.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="0 1">
            <marginalDist dimToTierMap="2" dims="1">1 2 3 5 8</marginalDist>
          </featureDist>
          <featureDist emptyComponentTiers="2">
            <marginalDist dimToTierMap="0" dims="1">1 1.2 1.5 2</marginalDist>
            <marginalDist dimToTierMap="1" dims="1">1 2 3</marginalDist>
          </featureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0 1 2" dims="3">1 2 3 4 4 2 3 3 4 5 2 3 3 4 5 3 4 5 6 6 4 5 5 6 7 4
5 5 6 7 6 7 8 9 9 7 8 8 9 10 7 8 8 9 10 11 12 13 14 14 12 13 13 14 15 12 13 13 14 15</marginalDist>
          </featureDist>
        </feature>
      </featureData>
      <combAvarData />
      <resourceEditData>
        <resourceEdit addressType="absolute" length="8" start="10000">
          <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="0" />
        </resourceEdit>
        <resourceEdit addressType="relative" length="8" start="2">
          <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="1" />
        </resourceEdit>
        <resourceEdit addressType="relative" length="8" start="4">
          <ssm:compVar compID="ssm:comp:myAudio" indType="layerInd" param="0" />
        </resourceEdit>
      </resourceEditData>
    </parcel>
    <parcel parcelID="1">>
      <componentData>
        <component compID="ssm:comp:myImage">
          <tierInfo numTiers="2">
            <tierElemInfo exclusiveFlag="false" numLayers="4" origLayers="4" tier="0" />
            <tierElemInfo exclusiveFlag="false" numLayers="4" origLayers="5" tier="1" />
          </tierInfo>
          <atomToc addressType="relative" start="23011">
            <atomTocEntry addressType="relative" indices="0 0" length="512" start="1000" />
            <atomTocEntry addressType="relative" indices="0 1" length="360" start="1600" />
            <atomTocEntry addressType="relative" indices="0 2" length="2000" start="2347" />
            <atomTocEntry addressType="relative" indices="0 3" length="2536" start="4400" />
            <atomTocEntry addressType="relative" indices="1 0" length="1006" start="7922" />
            <atomTocEntry addressType="relative" indices="1 1" length="1878" start="9042" />
            <atomTocEntry addressType="relative" indices="1 2" length="2663" start="10225" />
            <atomTocEntry addressType="relative" indices="1 3" length="3549" start="12922" />
            <atomTocEntry addressType="relative" indices="2 0" length="1470" start="19014" />
            <atomTocEntry addressType="relative" indices="2 1" length="2351" start="20765" />
            <atomTocEntry addressType="relative" indices="2 2" length="3534" start="23211" />
            <atomTocEntry addressType="relative" indices="2 3" length="4915" start="26819" />
            <atomTocEntry addressType="relative" indices="3 0" length="2029" start="32108" />
            <atomTocEntry addressType="relative" indices="3 1" length="3147" start="34137" />
            <atomTocEntry addressType="relative" indices="3 2" length="4258" start="37308" />
            <atomTocEntry addressType="relative" indices="3 3" length="5307" start="41566" />
          </atomToc>
        </component>
        <component compID="ssm:comp:myAudio">
          <tierInfo numTiers="1">
            <tierElemInfo exclusiveFlag="false" numLayers="4" origLayers="5" tier="0" />
          </tierInfo>
          <atomToc addressType="relative" start="49105">
            <atomTocEntry addressType="relative" indices="0" length="789" start="1178" />
            <atomTocEntry addressType="relative" indices="1" length="1745" start="1967" />
            <atomTocEntry addressType="relative" indices="2" length="2840" start="3825" />
            <atomTocEntry addressType="relative" indices="3" length="4173" start="6778" />
          </atomToc>
        </component>
      </componentData>
      <featureData>
        <feature avar="ssm:avar:imageResolution">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0" dims="1" distType="nonDecreasing">100 200 400
  800</marginalDist>
            <marginalDist dimToTierMap="1" dims="1" distType="constant">1.0</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:imageCodesize">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0 1" dims="2" distType="nonDecreasing">120 240 350 400 240
  300 420 510 315 456 600 720 378 526 720 823</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:imageDistortion">
          <components>
            <component compID="ssm:comp:myImage" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>2000.0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0 1" dims="2" distType="nonIncreasing">1400 1000 813 621
  1200 803 647 531 1000 671 436 317 699 475 355 206</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioCodesize">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>0</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0" dims="1" distType="nonDecreasing">105 211 323
  456</marginalDist>
          </featureDist>
        </feature>
        <feature avar="ssm:avar:audioDistortion">
          <components>
            <component compID="ssm:comp:myAudio" />
          </components>
          <refFeatureValue>1.0</refFeatureValue>
          <emptyFeatureDist>600</emptyFeatureDist>
          <featureDist emptyComponentTiers="">
            <marginalDist dimToTierMap="0" dims="1" distType="nonIncreasing">400 326 237
  160</marginalDist>
```

```
      </featureDist>
     </feature>
     <feature avar="ssm:avar:perceptualRichness">
      <components>
       <component compID="ssm:comp:myImage" />
       <component compID="ssm:comp:myAudio" />
      </components>
      <refFeatureValue>10.0</refFeatureValue>
      <emptyFeatureDist>0.0</emptyFeatureDist>
      <featureDist emptyComponentTiers="0 1">
       <marginalDist dimToTierMap="2" dims="1">2 3 5 8</marginalDist>
      </featureDist>
      <featureDist emptyComponentTiers="2">
       <marginalDist dimToTierMap="0" dims="1">1 1.2 1.5 2</marginalDist>
       <marginalDist dimToTierMap="1" dims="1">1 2 3 4 4.</marginalDist>
      </featureDist>
      <featureDist emptyComponentTiers="">
       <marginalDist dimToTierMap="0 1 2" dims="3">2 3 4 4 3 3 4 5 3 3 4 5 3 4 4 5 4 5 6 6 5 5 6 7 5 5
6 7 5 6 6 7 7 8 9 9 8 8 9 10 8 8 9 10 8 9 9 10 12 13 14 14 13 13 14 15 13 13 14 15 13 14 14
15</marginalDist>
      </featureDist>
     </feature>
    </featureData>
    <combAvarData />
    <resourceEditData>
     <resourceEdit addressType="absolute" length="8" start="62569">
      <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="0" />
     </resourceEdit>
     <resourceEdit addressType="relative" length="8" start="2">
      <ssm:compVar compID="ssm:comp:myImage" indType="layerInd" param="1" />
     </resourceEdit>
     <resourceEdit addressType="relative" length="8" start="10">
      <ssm:compVar compID="ssm:comp:myAudio" indType="layerInd" param="0" />
     </resourceEdit>
    </resourceEditData>
   </parcel>
  </parcelData>
  <globalFeatureData>
   <globalFeature avar="ssm:avar:imageResolution">
    <components>
     <component compID="ssm:comp:myImage" numLayers="5 5" />
    </components>
    <refFeatureValue>1.0</refFeatureValue>
    <emptyFeatureDist>0.0</emptyFeatureDist>
    <featureDist emptyComponentTiers="">
     <marginalDist dims="1" dimToTierMap="0" distType="nonDecreasing">100 200 400 800
1600</marginalDist>
     <marginalDist dims="1" dimToTierMap="1" distType="constant">1.0</marginalDist>
    </featureDist>
   </globalFeature>
  </globalFeatureData>
  <globalCombAvarData>
   <globalCombAvar avar="ssm:avar:codesize">
    <ssm:adapVar avar="ssm:avar:audioCodesize" />
    <ssm:adapVar avar="ssm:avar:imageCodesize" />
    <ssm:operation operator="add" />
  </globalCombAvar>
   <globalCombAvar avar="ssm:avar:lagrangian" numArguments="2">
    <ssm:adapVar avar="ssm:avar:audioDistortion" />
    <ssm:adapVar avar="ssm:avar:imageDistortion" />
    <ssm:argument number="0" />
    <ssm:operation operator="multiply" />
    <ssm:operation operator="add" />
    <ssm:adapVar avar="ssm:avar:codesize" />
    <ssm:argument number="1" />
    <ssm:operation operator="multiply" />
    <ssm:operation operator="add" />
   </globalCombAvar>
   <globalCombAvar avar="ssm:avar:myImageImpliesAudio">
    <ssm:compVar compID="ssm:comp:myAudio" indType="inclusionInd" />
    <ssm:compVar compID="ssm:comp:myImage" indType="inclusionInd" />
    <ssm:operation operator="boolNOT" />
    <ssm:operation operator="boolOR" />
   </globalCombAvar>
  </globalCombAvarData>
  <globalLUTAvarData>
   <globalLUTAvar avar="ssm:avar:lagrangianLUT " numAxes="2">
    <axisValues axis="0" grid="25 50 100" />
    <axisValues axis="1" grid="10 30 100 300" />
    <content>
     <marginalDist dims="2" dimToTierMap="0 1">0.9 0.8 0.7 0.6 0.5 0.4 0.8 0.7 0.6 0.5 0.4 0.3 0.7 0.6
0.5 0.4 0.3 0.2 0.6 0.5 0.4 0.3 0.2 0.1 0.5 0.4 0.3 0.2 0.1 0.1</marginalDist>
    </content>
   </globalLUTAvar>
  </globalLUTAvarData>
  <globalCreatorLimitConstraints>
   <globalLimitConstraint highLimit="1" lowLimit="1">
    <ssm:adapVar avar="ssm:avar:myImageImpliesAudio" />
   </globalLimitConstraint>
  </globalCreatorLimitConstraints>
  <codecOffsetData>
   <offsetReference addressType="relative" start="34508">
    <offsetEntry addressType="relative" invalidPointerHandling="moveUp" length="16" start="2"
value="10000" />
    <offsetEntry addressType="relative" invalidPointerHandling="moveUp" length="16" start="4"
value="20000" />
   </offsetReference>
   <offsetReference addressType="relative" start="3000">
    <offsetEntry addressType="relative" invalidPointerHandling="moveUp" length="20" start="2"
value="-6178" />
    <offsetEntry addressType="relative" bitPos="4" invalidPointerHandling="moveUp" length="20"
start="4" value="23011" />
    <offsetEntry addressType="relative" invalidPointerHandling="moveUp" length="20" start="7"
value="39933" />
   </offsetReference>
  </codecOffsetData>
  <sequenceData>
   <sequence startValue="10" stepValue="2">
    <countField addressType="absolute" length="16" start="10000" />
    <countField addressType="relative" length="16" start="5000" />
    <countField addressType="relative" length="16" start="1000" />
    <countField addressType="relative" length="16" start="2000" />
   </sequence>
   <sequence startValue="0" stepValue="1">
    <countField addressType="absolute" length="8" start="20386" write="false" />
    <writeField addressType="absolute" length="8" start="25837" />
   </sequence>
   <sequence startValue="0" stepValue="6">
    <writeField addressType="absolute" length="8" start="29223" type="seqValue" />
    <subSequence pack="false" startValue="0" stepValue="1">
     <countField addressType="relative" length="8" start="1000" write="true" />
     <countField addressType="relative" length="8" start="1000" write="true" />
     <countOnly />
     <countOnly />
     <countOnly />
     <countOnly />
    </subSequence>
    <subSequence modulo="8" pack="false" startValue="0" stepValue="1">
     <writeField addressType="absolute" length="8" start="73441" type="count" />
    </subSequence>
   </sequence>
  </sequenceData>
</SSMDescription>
```