White Paper on Rich Media Environment Technology Landscape Report

Candidate – 14 Oct 2008

Open Mobile Alliance

OMA-WP-Rich_Media_Environment-20081014-C

Contents

Figures

Tables

1 Scope

The objective of this white-paper document is to provide technical descriptions and initial analysis of available technologies that are candidate to the RME enabler specification, answering the RME requirements. Note that Sections 5, 6, and 7 describe the analysis of the technogies by the proponents and a detailed analysis by OMA will be done in the next phase of the specification.

In the Rich-Media Environment, a Rich Media service is a dynamic, interactive collection of multimedia data such as audio, video, graphics, images and text. It ranges from a movie enriched with vector graphics overlays and interactivity (possibly enhanced with closed captions), to complex multi-step services with fluid interaction/interactivity and different media types at each step. The demand for such Rich Media service is increasing at a high pace, spurred by the development of the next generation mobile infrastructure and the generalization of TV content to new mobile environments.

As a consequence the scope of the RME focuses on dynamic rich-media services, where the services offered to the end-user are enhanced by the cooperative linkage between media (e.g.: synchronisation between events and media, real-time delivery of content…) combined with interactivity mechanisms and/or end-user interaction. The RME enabler can be use as a generic enabler, allowing creating dynamic interactive rich-media services and can also benefit, or be used in association with other OMA enablers such as BCAST or DCD.

As the RME enabler is bearer agnostic, its functionalities shall not be restricted to or by the usage of a particular bearer.

The RME enabler complements the browsing enhancement work item, covering “web on mobile”, “web browsing” or “web application” in which the navigation and visualisation mode is on a page by page basis and where in general, the dynamic data are not rich-media (for instance in a flight booking service), where modification needs an user action (as opposed to streamed data) and where the timing model does not depend on real-time data nor implies tight synchronisation.

To achieve this goal the RME WP will:

- List available technology (e.g.: based on the RME WID)
- Provide a technology landscape
- Analyse technology against requirements
- Open the discussion to specify the RME enabler in the RME TS.

2 References

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. November 1997, URL:http://www.ietf.org/rfc/rfc2234.txt

3 Terminology and Conventions

3.1 Conventions

This is an informative document, which is not intended to provide testable requirements to implementations.

3.2 Definitions

N/A

3.3 Abbreviations

BIFS	Binary Format For Scene
LASeR	Lightweight Application Scene Representation
MPEG	Moving Picture Expert Group
OMA	Open Mobile Alliance
SAF	Simple Aggregation Format
SVG	Scalable Vector Graphic
W3C	World Wide Web Consortium

4 Introduction

This white-paper document provides technical descriptions and initial analysis of available/proposed candidate technologies for the RME enabler specification to address the RME requirements.

In the Rich-Media Environment, a Rich Media service is a dynamic, interactive collection of multimedia data such as audio, video, graphics, images and text. It ranges from a movie enriched with vector graphics overlays and interactivity (possibly enhanced with closed captions), to complex multi-step services with fluid interaction/interactivity and different media types at each step. The demand for such Rich Media service is increasing at a high pace, spurred by the development of the next generation mobile infrastructure and the generalization of TV content to new mobile environments.

As a consequence the scope of the RME focuses on dynamic rich-media services, where the services offered to the end-user are enhanced by the cooperative linkage between media (e.g.: synchronisation between events and media, real-time delivery of content…) combined with interactivity mechanisms and/or end-user interaction. The RME enabler can be use as a generic enabler, allowing creating dynamic interactive rich-media services and can also benefit, or be used in association with other OMA enablers such as BCAST or DCD.

As the RME enabler is bearer agnostic, its functionalities shall not be restricted to or by the usage of a particular bearer.

The RME enabler is intended to complement the OMA browsing enhancement work item, covering “web on mobile”, “web browsing” or “web application” in which the navigation and visualisation mode is on a page by page basis and where in general, the dynamic data are not rich-media (for instance in a flight booking service), where modification needs an user action (as opposed to streamed data) and where the timing model does not depend on real-time data nor implies tight synchronisation. While technology solutions for the browsing enhancements work item are not specifically called out as a solution for RME the ability to integrate with the browser, especially in meeting the defined requirements.

5 Potential Technologies for RME

The following is the list of technologies or technology combinations proposed for consideration to meet the requirements defined in the requirements document [RME-RD].

- MPEG4 part 20 (LASeR) [LASeR]
- MORE

5.1 MPEG4 part 20 (LASeR)

LASeR, formally known as ISO/IEC 14496-20 (MPEG-4 Part 20), is the new Rich Media standard dedicated to the mobile, embedded and consumer electronics industries specified by the MPEG standardization group. The objective of MPEG4 part20 are to enable a fresh and active user experience on constrained networks and devices based on enriched content, including Audio, Video, Text, and Graphics And to addresses the requirements of the end-to-end rich media publication chain: ease of content creation, optimized rich media data and streams delivery and enhanced rendering on all devices.

MPEG4 part 20 is composed of LASeR (Lightweight Application Scene Representation) and SAF (Simple Aggregation Format).

LASeR is a format for scene description and:

- Is inspired by the concepts of Macromedia Flash™, and ISO/IEC MPEG/BIFS,
- Is based on W3C/SVG specification; LASeR cumulates the particular knowledge and know-how of both W3C and MPEG groups.

LASeRv2 supports:

- The SVG Tiny 1.2 scene description
- A set of key compatible extensions over SVG Tiny1.2,
- The ability to encode and transmit LASeR files, LASeR scene fragments, and private XML data.
- The ability to encode and transmit LASeR stream. LASeR content can be delivered into packaged pieces, allowing display as soon one piece is received (as opposed to a download an play mechanism). This concept of “streaming”, already into place for audio and video data, has been generalized to scene description and Rich Media. As such, services can be designed such that there is always some information of interest on the screen.
- Dynamic updating of the scene to achieve a reactive, smooth and continuous service.
- Simple yet efficient compression to improve delivery and parsing times, as well as storage size.
- An efficient interface with audio and visual streams with frame-accurate synchronization.
- Usage of any font format.
- Easy conversion from other popular rich-media formats in order to leverage existing content and developer communities.

LASeR v2 is a super-set of LASeR v1. See Section 7.1.1.

SAF aims at fulfilling all the requirements of rich-media services at the interface between media/scene description and existing transport protocols:

- Simple aggregation of any type of stream, file or fragment.
- Dynamic addition of new streams/files after the start of the delivery.
- Media interleaving,
- Precise synchronization mechanisms support.
- Signalling of MPEG and non-MPEG streams,

- Optimized packet headers for bandwidth-limited networks to guarantee a very low overhead,
- Easy mapping to popular streaming formats,
- Enhanced support for progressive download.
- Real time transmission/delivery.
- Cache management capability.
- Extensibility such as adding new packet types or new stream types.

LASeR and SAF can be used independently.

Availability of the specifications:

- LASeR v1 has reached the FDIS stage in October 2005.
- LASeR v2 will reached the FPDA stage in October 2006.

5.2 The Mobile Open Rich-media Environment (MORE)

MORE is an open suite of W3C, OMA, 3GPP and IETF technologies combined to meet the requirements for formatting, packaging, compressing, transporting, rendering and interacting with rich media files and streams. The system leverages the respective components of existing W3C, 3GPP, MPEG, OMA solutions such as the SVG Mobile 1.2 Profile, MBMS, ISO Media File Format and browsing enablers. MORE solution is also compatible with JSR-226, which defines an API for loading and manipulating SVG Tiny 1.1 content using a compatible Micro DOM subset tailored for Mobile Java Environment.

The scene update syntax in MORE will rely on the REX (Remote Events for XML) initiative in W3C that is spear-headed by SVG WG in an effort to meet the requirements of RME/DIMS specifications. The proposed XML update specification will be based on a set of requirements that are intended to maintain compatibility with DOM events, declarative in nature, and integrates well with the WWW architecture. The current charter of the Web Applications API will be responsible for maintaining this specification. Note that the syntax for update mechanism is not limited only to SVG but also extensible to other mark-ups, besides being very efficient and light weight for platforms that are already capable of supporting mobile SVG standard.

As the underlying presentation format for rich media in both OMA-RME and 3GPP-DIMS work-items is SVG based, MORE provides a solution to embed vector graphics content such as SVG into the existing 3GPP ISO Base Media File Format for streaming of live rich media content over MMS/PSS/MBMS services. This method will allow the container format to be used for packaging rich media content (graphics, video, text, and images), enabling streaming servers to generate RTP packets, and clients to interact, realize, play, or render rich media content.

MORE provides the ability to support interaction among the rich media clients and servers. Mechanisms for interactivity include provisions for local (client side) and remote interaction (server-client), as well as for real time and non-real time feedback over various broadcast and peer-to-peer transport protocols. Local interaction mechanisms in MORE are based on SVG Mobile 1.2 event model, designed after the W3C XML events and DOM Level 3 Events model. For remote interaction, MORE provides a framework and message format syntax for client feedback.

The following sub-sections presents an overview of the components involved in the Rich Media system architecture. It also provides references to related activities that are taking place in other Standards Organizations (SDOs) that MORE would like to collaborate closely with and utilize in order to exploit the synergies that exist between them.

5.2.1 Scene Presentation Format

This component refers to scene presentation format for compositing, document language processing and rendering rich media (vector and raster graphics, audio, video, text) content, providing backward and forward compatibility with open standards specifications including the compatibility with uDOM API, dynamic updating of rich media content, being able to interface with the open standards based browser client and providing referencing and synchronization among the different media. MORE uses SVG Mobile 1.2 (also referred to as SVG Tiny 1.2) as a basis for its presentation format without profiling. Any

necessary extensions will be defined in an extensible and compatible manner. Since SVG Mobile 1.2 is developed in W3C, this work should involve close collaboration with them for achieving interoperability at the highest level.

5.2.2 Scene Update Format

This component refers to scene update format and mechanisms. MORE will rely on the REX (Remote Events for XML) initiative, a joint effort between W3C SVG and Web Apps working groups (http://www.w3.org/TR/rex/). The proposed XML update specification will be based on a set of requirements that are intended to maintain compatibility with DOM events, declarative in nature, and integrates well with the WWW architecture and the general requirements of the WAE (Wireless Applications Environment).

5.2.3 Local User Interaction

This component refers to the ability to allow the user to interact locally with the rich media content in both real- time and as well as non-real time. It also includes event handling with the dynamic updates of the rich media content i.e. generation of DOM Mutation events as a result of applying scene updates. These are seen as application level protocols such as scripting and interaction with DOM. Local event management should be handled through W3C DOM Events. As the expertise for integrating this area lies in OMA, this work will be defined in OMA with necessary alignment with W3C. To ensure interoperability on the client, JCP is also relevant as the DOM definitions are common between W3C SVG Mobile 1.2 and JCP JSR-226 specifications.

5.2.4 Container/Delivery Format

This component refers to the packaging of different rich media data into a file that can be used as a storage format for download, progressive download and streaming profiles. MORE proposes using existing ISO Base Media File Format that is used extensively in mobile services today, therefore it is preferred that we work with 3GPP in this area as much as possible.

5.2.5 Re-synchronization and Tune-in

This component refers to the client's ability to be able to re-synchronize with and tune-in to the rich media service. MORE provides several mechanisms to aid this purpose such as the usage of random-access points, transmission of current list of active elements, and the transmission of scene information at key intervals. This component has implications on both application and underlying transport, therefore MORE proposes to coordinate closely with 3GPP as much as possible when dealing with synchronization at the transport level.

5.2.6 Transport

This component refers to the transport mechanisms for delivering rich media content to the terminal including the ability to support download and play, progressive download and real-time streaming, efficient transmission of rich media data and updates over standard protocols such as HTTP, FLUTE, and RTP. MORE facilitates random access to different parts of the presentation over time, graceful handling of packet loss, retransmission and error correction of lost packets, These lower-level transport technologies are outside the direct scope for OMA; MORE solution proposes to coordinate any work in this area with 3GPP/IETF as needed.

5.2.7 Compression

The use of compression and content specific encoding techniques are economically driven decisions. Rich media content consists of SVG scenes and scene updates along with other referenced media. For streaming purposes, existing compression methods can be used for referenced media. However, compressing small sized SVG does not yield high benefits with the available bandwidth in today's networks. For large content, MORE recommends using Gzip as it results in high compression ratio. Hence, there is no specific need for introducing a new compression mechanism for rich media. Note however, that MORE does not preclude application of a specific encoding scheme that is widely adopted in the industry. This approach may be modified depending upon the outcome of the W3C work on XML compression as it tries to address compression for arbitrary XML data and not schema specific. In any case, it is important to view any encoding and compression decisions as orthogonal and separable from any base design decisions

The following table presents the different components or scope areas within the MORE architecture, brief description, relevant SDOs and their publication status.

Component/ Scope Area	Description	SDO	Reference
Scene Presentation Format	Rich Media Scene representation format is based on SVG Mobile 1.2 - Extensible to CDF in the future	W3C/OMA	W3C SVG Mobile 1.2 *Publication Status*: W3C Last Call Working Draft Estimated Time for Completion: CR-April 2006
Scene Update Format	Scene update syntax based on DOM Event based processing model. Remote Events for XML (REX)	W3C/OMA	W3C REX *Publication Status*: W3C Public Working Draft Estimated Time for Completion: CR-June 2006
Local User Interaction	DOM Level 3 Events Model (user events, DOM events)	W3C/OMA	W3C SVG Mobile 1.2/DOM Level 3 Events *Publication Status*: W3C Note
Container/Delivery Format	3GP extensions (new track for SVG content, etc.) for download and streaming	3GPP/OMA	MORE proposal *Publication Status*: Member Proposal Estimate Time for Completion: Dependent on RME timeline
Re-synchronization & Tune In	Random access points, tune-in	3GPP/OMA	MORE proposal *Publication Status*: Member Proposal Estimate Time for Completion: Dependent on RME timeline
Transport	Payload format and types for SVG content for RTP streaming	3GPP/OMA	MORE proposal *Publication Status*: Member Estimate Time for Completion: Dependent on RME timeline Proposal
Compression	GZIP	IETF	RFC 1952 *Publication Status:* Final

Table 1: MORE components

6 Comparision of the technologies against the requirements

In this section the technologies are simply compared against the requirements defined in [RME-RD] without any form of judgement as to suitability being applied. The requirements are grouped into themes with the detailed requirements being subsidiary to the themes. A tabular means is used to allow easy comparison.

In this table the supports column concerns what is standardised in existing specifications used or re-used by candidates. Application or final solution based on one or the other candidates will fulfilled all the requirements by specifying the "missing part" if needed. Some of the requirements can only be fulfilled at the implementation level.

The labels in the table are defined as follow:

'Yes' - The requirement is met by already defined and approved specification.

'Being specified' -The requirement is met by the specification that is being developed in a specific release and indicative timescale exists. This requires the feature is already been agreed by the group developing the specification.

'Proposed' - The requirement can be met by a proposal submitted to the group developing the specification but there is no commitment by the owning organisation to develop the feature.

A proposal shall be provided with enough information so that we can assess it may meet the requirement.

The ownership of the submitted proposal shall be indicated (e.g;: W3C, 3GPP, OMA, MPEG, IETF…).

'No' - None of above. The requirement is out-of-scope of the candidate.

Requirement Groups and detailed requirements		MPEG 4 part 20		MORE	
		Supports	**Comments**	**Supports**	**Comments**
General Requirement for the media-type	RME-FUNC-001 The RM enabler SHALL support methods to minimize the latency perceived by the end user.	Yes	LASeR provides multiples features to achieve this goal, in particular: 1 - a binary format allowing a fast parsing speed, a fast delivery of highly compressed content, 2 - a very efficient declarative dynamic update mechanism to have always modifications on the end-user screen and to replace the page by page navigation provided by XML based technology. 3 – means to play content while waiting feedback from a request, while buffering data or while waiting for a new scene. LASeR v2 provides enhanced media management for the mobile environment ("StreamSource"), especially addressing the specificities of broadcasting	Yes Being Specified Proposal (3GPP)	MORE supports the following methods to minimize perceived latency. 1) MORE provides for progressive download/rendering mechanisms as specified in SVG Mobile 1.2 specification. 2) MORE uses GZIP as the compression mechanism. GZIP can be used for large scenes and small sized content can be transmitted as is to reduce overhead.
		Being specified			

		networks, allowing for low-latency channel switching. Detailed information in section 7.1		3) Supports dynamic updates of content. (REX) 4) Transport of dynamic updates Please refer to transport section in the MORE proposal.
RME-FUNC-002 It SHALL be possible to present multiple RM data sources within a single scene.	Yes	The design of the LASeR engine supports the integration within one scene of data and streams coming from various origins, e.g. different HTTP servers, DVBH + 3G, HTTP + RTP, etc. This can be achieved through the 'href' attribute or through StreamHeader and RemoteStreamHeader units..	Yes	In MORE, the scenes allow for embedding raster and vector graphics, video and audio with SVG as defined in the SVG Mobile 1.2 specification. With SVG, one can embed (base64) and reference (xlink:href) media. In addition, using xlink:href, one has the ability to reference data from multiple external resources.
RME-FUNC-003 The RM enabler SHALL be able to render, within one scene, data and updates received from different sources (eg: networks and delivery mechanisms, content provider). Note: the service provider should be the same	Yes	The design of the LASeR engine supports the integration within one scene of data and streams coming from various bearers. E.g.: DVB-H + 3G networks This can be achieved through the 'href' attribute or through StreamHeader and RemoteStreamHeader units.	Yes Proposal (3GPP/OMA)	The scene updates allow for embedding raster and vector graphics, video and audio with SVG as defined in the SVG Mobile 1.2 specification. For timing, the RM enabler makes use of run-time synchronization functionality that SVG Mobile 1.2 inherits from SMIL 2.0. These attributes are syncBehavior, syncTolerance and syncMaster attributes, specified on the 'audio', 'video' and 'animation' elements, and syncBehaviorDefault and syncToleranceDefault attributes specified on the svg element. For resolving content timing conflict (e.g.

				one source attempts to add an SVG element, and another source attempts to delete the SVG element), session priority is applied through the Session Description Protocol (SDP). The terminal takes care of this priority information at its discretion. To resolve updates received from different sources, timing and inter source conflict have to be taken into consideration.
RME-FUNC-015 The RM enabler navigation and interaction SHALL be agnostic to the type of MMI provided (eg using any input device)	Yes	The LASeR specification provides specific tools to ease the authoring of services which are independent from the type of MMI on the phones. In particular LASeR allows the emulation of a pointing device (stylus or mouse) through the use of a keyboard, thus allowing content to be designed in an input-device independent manner.	Yes	SVG Mobile 1.2 content specification is agnostic to MMI. MORE recommends that the author utilizes input device agnostic events. For example, designing content to react to the “activate” UIEvent rather than a “click”. MouseEvent.
RME-FUNC-017 The content creator or the service provider SHALL be able to define the lifecycle of RM data.	Yes	LASeR specifies means to attribute at the content/service creation the availability and the timing of any kind of RM data. LASeR provides tools to manipulate the scene content by adding, modifying and removing scene elements. Data discarding is possible with any kind of service logic: on a frame-accurate basis, on a relative time, on end-user action, on end-user navigation, on an absolute time, etc.	Yes Being Specified	MORE re-uses the definitions of the start time, end time, duration with ‘begin’, ‘dur’, and ‘end’ timing attributes respectively, as defined by SVG Mobile 1.2 profile. These attributes can be specified for the SVG data as well as the embedded media. For scene and scene updates the start time is relative to the presentation time. Also, the SVG `discard’ element can be used on specific elements to denote when they need to be discarded from memory. It is also possible to use the DOMNodeRemoved

				event in REX to transmit a server event that discards useless data.
RME-FUNC-024: The RM enabler SHALL be codec, network, terminal, browser, middleware, OS agnostic.	Yes	LASeR engines can interface with any kind of media type. The LASeR specification is codec network terminal browser, middleware and OS agnostic.	Yes Proposal (3GPP/OMA)	MORE is compliant with existing open standards technologies and therefore agnostic to terminal, browser, middleware, and OS. For codecs, MORE recommends to use the codecs specific to the service (for e.g. 3GPP, 3GPP2, etc.) specifications.
RME-FUNC-025 The RM enabler SHALL allow an end to end optimizations to be applied (eg: compression, preparsing, data preconditionning)	Yes	LASeR provides both an highly efficient compression mechanism a preconditioning mechanism and a packaging of data by using SAF Furthermore a LASeR content is always provided as a complete and well formed fragment/packet whatever are the transmission modes, for checking, reliable and optimisations purposes. See § 7.1.4.1 and 7.1.4.2 in Section 7.	Yes	For streaming purposes, existing compression methods can be used for embedded media (e.g. audio, video, images). Any 3GPP/OMA supported media codecs can be used. For example, AMR Wideband for audio, H.264 for video, etc. For compressing scene content ubiquitous GZIP is used. MORE does not mandate a specialized encoding method.
RME-FUNC-026 RME-USA-001 The RM enabler functionality SHOULD be scalable from constrained terminals to unconstrained terminals.	Yes	LASeR allows implementations from medium-range J2ME devices (MIDP1/2) to higher-end PDA-like devices. By definition, the Binary format specified by LASeR provides a small footprint of the LASeR User agent suitable for both constraint and unconstraint terminal	Yes	MORE, through the use of SVG Mobile 1.2, allows for designing content that are optimized for both very constrained and unconstrained terminals. This is achieved by using SVG feature strings that allows for multiple execution paths within the same content based on client capabilities. E.g. the same content may put a rotation on a video for high-end devices and use the same video axis-aligned on low-end devices.

				In other words content designed for constrained terminals is rendered identical on unconstrained terminals
RME-FUNC-027 It SHALL be possible to invoke an external application (eg MMS SMS clients) from within the service and it MAY be possible to integrate those applications visually within the service interface.	No	LASeR engines can be interfaced with external applications	No	MORE does not preclude this functionality. MORE intends to align with the current OMA architecture.
RME-FUNC-028 Personalisation by the end-user of content, delivered within the service SHALL be possible.	Yes Being specified	LASeR can do this through LASeR Commands Save, Restore and Clean and do not require to reload or re download a complete content / application. The LASeR update mechanism is generic, can apply to any XML data and can be implemented easily on top of uDOM.	Yes Being Specified	Based on the preferences, the application may chose to alter the content using the uDOM and scripts. The alternative method is to send these changes via updates (REX) from the service provider.
RME-FUNC-029 : Text scrolling and slideshow SHALL be provided	Yes/ Being specified	LASeR v1 has the same functionality for text scrolling and slide show as SVGT1.2 LASeR v2 is specifying in addition another tool to achieve more functionality for text scrolling e.g. management of dynamically changing text, unknown font size...	Yes	The MORE client allows for scrolling and rendering slideshows with the use SVG Mobile 1.2 features. For e.g. The animation element controlling the scrolling text can be updated when the text itself is updated using a script that queries the text width.
RME-FUNC-030 The RM enabler SHALL allow best effort font management regardless of screen size language and fontstyle.	Yes/	The design of the LASeR client does not mandate any font systems . LASeR recommends the usage of OpenType font system that provides a convenient hinting of characters for any language and in particular for Arabic or Asian language LASeR specifies that font can be encoded within the content using	Yes	MORE supports both system fonts as well as SVG based embedded fonts. The MORE proposal does not place any restrictions on the use of text and fonts in content and therefore provides the same level

			the anyXML attribute. LASeR specifies that font can be transported as separate streams including SVG font encoded as opentype font. LASeR allows the use of system fonts and specifies a fallback mechanism for that. LASeR v2 will provide support of SVG font.		of support available in SVG Mobile 1.2.
		Being specified			
	RME-SEB-001 The RM Enabler SHOULD be able to reference and to access AV Streams	Yes	LASeR inherits this feature from SVG Tiny 1.2. LASeR benefits from the MPEG decoder model and synchronisation support to complement this feature.	Yes	MORE allows for the inclusion of internally embedded media as well as the referencing of such externally linked media. It can be done at both media type.
	RME-USA-003 The RM Enabler SHOULD continue to render the RM service while content requested by the end user is not yet available (triggered by a click on an url, or a press on a key).	Yes/ Being specified	The LASeR design allows to maintain a continuity of rendering while requesting content. This feature masks the perceived latency. LASeR V2 provides an additional media management for mobile environment ("StreamSource"), answering to this requirement for A/V sources.	Yes	SVG Mobile 1.2 supports for rendering content while requesting for resources currently unavailable. The user can request for resources through scripting. The user can also fetch the external resources similar to resources obtained by using by xlink:href.
	RME-USA-002 The RM Enabler SHOULD use a very small footprint and require very limited performance when using the smaller sets of features.	Yes	By using a binary format for the rich-media data, LASeR allows smaller and faster implementations with a smaller memory footprint. For information the LASeR Reference Software (Jar file) in Java MIDP2, non optimised, is around 100kBytes See description of the reference software in § 7.1.9.2.	Yes	MORE is based on SVG Mobile 1.2 profile, which is designed for small footprint. As MORE is based on SVG Mobile 1.2, it results in substantial savings of memory footprint since a RME enabler can reuse the SVG engine that in most cases already will be present on the device and used within several other enablers (browser, MMS etc.).
Interaction requirements	RME-FUNC-013 It SHALL be possible for the RM enabler to Interact with the	Yes	The backchannel is the same with SVG Tiny 1.2 and LASeR, but the response is much more efficient when: Content is binary encoded	Proposal	MORE supports remote interaction between the enabler and source of the content by extending existing

	source of the rich-media content		Using the LASeR Commands to answer the user request. Using the append mode See § 7.1.7.2		protocols such as SMS, MMS, HTTP and RTSP. Note that MORE considers these different protocols (rather than extends) only to provide flexibility of protocol usage for remote interaction.
	RME-FUNC-016 The RM enabler SHALL be able to discard RM data when it has been identified as no longer useful in the service.	Yes	The discard of data is possible with any kind of service logic: on a frame-accurate basis, on a relative time, on end-user action, on end-user navigation, on an absolute time, etc. See 7.1.2.1.5 and 7.1.5	Yes Being Specified	MORE uses the 'discard' element in SVG Mobile 1.2 allows authors to specify the time at which particular elements may be discarded. This is particularly useful for enablers to handle long-running documents. The attribute can be either set during the content creation time or the during the user interaction with the help of scripting. Or the service provider may wish to use the DOMNodeRemoved event in REX to transmit a server event that discards useless data.
	RME-USA-003.1 The author SHOULD have the choice of specifying what should happen between the request and the arrival of the content or during buffering. Eg: continue to play the current scene, play a specific pre-buffered animation or transition, or do nothing.	Yes	The LASeR design allows the rendering of either the current content, or another content while waiting for requesting content. The latency is masked by this capability	Yes	MORE re-uses the <prefetch> functionality as specified in SVG Mobile 1.2. Prefetch functionality is used for the user agent to specify how the content needs to be buffered before a smooth presentation can be rendered
Timing	RME-FUNC-006 RM data	Yes	MPEG4 part 20 provides tools to manage timed based services and	Yes	At the application level, MORE uses the

and synchronisation requirements for the media-type	rendering time and synchronisation SHALL be controllable by the RM enabler.		frame based services. In the LASeR scene format the rendering time is controllable in conformance with content creator wishes. If a content is designed for 15 frames per second and the devices support a 10 frames per second display: the choice of the rendering policy belongs to the content provider: either to have a long experience but all frame are displayed, or a timed experience, and frame are dropped. Synchronisation between time-based media and static media or between multiple time-based media is possible. LASeR allows the creation of frame-based content, and provides means to establish interactive links from/to any frame, based on a timed logic.	Proposal (3GPP)	timing attributes as specified within the SVG language along with the attributes for run-time synchronization can be used to precisely control synchronization between frame-based and non-frame based media. At the transport level, MORE provides timing and time to decode information, stored in the ISO Base Media File Format to synchronize samples of different media. MORE also uses the RTP timestamp in the RTP payload (taken from the media time sample boxes in the container format) and the NTP timestamp (in the RTCP sender report) to form a pair that identifies the absolute time of a particular sample in the stream.
	RME-FUNC-007 The service provider SHALL be able to express an appropriately accurate synchronisation for the RM data which SHOULD be honoured by the enabler.	Yes	LASeR inherits from SMIL and SVG Tiny 1.2 the means to signal the author-specified synchronisation request. LASeR inherits from MPEG-4 Systems a timing model allowing the scene updates to be synchronized with other streams. LASeR defines precisely how to recover the scene time information from the transport time stamp in order to benefit fully from the synchronisation support offered by transport layers As a result of all the above, LASeR allows the implementation of a complete synchronization framework which works on scene and media streams in a unified manner.	Yes Proposed (3GPP)	MORE provides this information in multiple ways: 1) As specified in SVG Mobile 1.2. 2) Through the ISO Base Media File Format. Also see above response to FUNC-006.
	RME-FUNC-008 Progressive rendering of RM	Yes	LASeR inherits progressive rendering from SVG, but also offers significantly improved means to	Yes	Supports this feature as described by SVG Mobile 1.2

data SHALL be provided		achieve streamability, through the use of LASeR Commands. See § 7.1.4.2		Specification (Ref: Chapter 5.9.2)
RME-FUNC-009 RM content SHALL be dynamically updatable in real time by the RM enabler	Yes	LASeR updates are "object based", declarative, simple and efficient to implement, use an easy-to-parse binary format and cover the whole scope of tree modifications. LASeR updates do not rely on scripting. The append mode and the MPEG timing model also contribute to the power of the MPEG updates. See laser specification § 6.4	Being Specified Proposed (3GPP/OMA)	MORE relies on the REX (Remote Events for XML) initiative, a joint effort between W3C SVG and Web Apps working groups. The proposed XML update specification will be based on a set of requirements that are intended to maintain compatibility with DOM events, declarative in nature, and integrates well with the WWW architecture. Scene updates in the form of add, delete, replace operations based on DOM events and can be streamed at real time to the client.
RME-FUNC-012 The service provider SHALL be able to create links between RM content at arbitrary times or places in the scene	Yes	Any pixel can be addressed as an interactive point within the scene. In addition to the means to create links inherited from SVG Tiny 1.2, LASeR allows the creation of frame-based content, and provides means to establish interactive links from/to any frame, based on a timed logic.	Yes Proposal (3GPP)	Supports linking between content as specified in SVGT 1.2 using xlink:href attribute and Animation module. Note that SVG provides the capability of seeking between arbitrary times both by using the xlink:href property when pointing to an animation and by setting the currentTime attribute in the uDOM. In addition, random access is provided to create links to arbitrary points in the scene/content. Further, the timing for each sample is specified in the ISO Base Media File Format, and is present as a timestamp for the RTP packets formed out of this container format.

RME-FUNC-014 Interactivity and interaction SHALL be possible on a frame accurate basis (time code or relative time).	Yes	The interactivity and the interaction are possible within a stream of rich-media data, allowing real time and precise synchronization of the interaction. For non-streaming delivery of rich-media data the synchronization is also provided on a frame-accurate basis.	Proposal (3GPP)	MORE provides interaction functionality at greater precision at sync samples as well as millisecond level to allow for greater time accuracy. Sync Sample Box and Shadow Sync Sample Box are defined in ISO Base Media File Format. The Sync Sample Box provides a compact marking of the random access points within the stream. If the sync sample box is not present, every sample is a random access point. For more details please refer to the detailed proposal.
RME-REL-001.1 The RM enabler SHALL be able to support re synchronisation with an existing active stream.	Yes	LASeR provides means to support carrousseling and means to synchronise scene and rich-media content with time-based media.	Proposal (3GPP)	MORE allows provision for quick tune in to an existing active stream during the presentation. The following mechanisms are used: - Random access points - Transmitting the current scene in short intervals to the tuned in client. - The use of timing for packet ordering and packet expiration.
RME-REL-001.2 The RM enabler SHALL support arbitrary access points to tune in the middle of content	Yes	LASeR provides the RefreshScene mechanism which is designed to provide tune-in capabilities in a broadcast like environment, as well as error recovery in a lossy streaming environment.	Proposal (3GPP)	MORE provides this information through the container format, which provides random access points to allow clients to tune into or access an arbitrary random access point in the presentation.
RME-SEB-003 The RM Enabler SHALL be able to specify multiple synchronisation masters. (E.g.:	Yes	This functionality is provided both for broadcast consideration usage (e.g.: mosaic menu) and for situations with multiple interactively triggered media within one scene.	Yes	MORE supports synchronization of different media at the application level, MORE utilizes the run-time synchronization

	This is required to deal with situations dealing with multiple synchronized groups of streams, such as video-on-demand.)				functionality that SVG Mobile 1.2 inherits from SMIL. These attributes are syncBehavior, syncTolerance and syncMaster attributes, specified on the 'audio', 'video' and 'animation' elements, and syncBehaviorDefault and syncToleranceDefault attributes specified on the svg element.
Reliability requirements for the media-type	RME –REL-001 The RM enabler SHALL support graceful handling of packet loss.	Yes	SAF provides a way to detect packet loss and LASeR is designed to handle gracefully incomplete scenes. LASeR uses the same SVGT1.2 error handling model and specifies an error handling process for the updates.	Proposal (3GPP/OMA)	MORE also supports graceful recovery from packet loss at the media level by providing mechanisms for error detection and error concealment as described in the MORE detailed proposal.
	RME-REL-001.3 The RM Enabler SHOULD handle duplicated data provided for error recovery purposes.	Yes	LASeR provides the RefreshScene mechanism which is designed to provide tune-in capabilities in a broadcast like environment, as well as error recovery in a lossy streaming environment	Proposal (3GPP/OMA)	MORE provides solution for error recovery requirement through determining the level of scene dissimilarity information. Based on this information the user agent can chose to either replace/refresh or not. Note that error recovery is relevant to both network and media type.
	RME-IOP-001 Newer versions of the RM enabler SHALL be backward compatible	Yes	LASeR inherits from SVG Tiny a provision for versioning so that LASeR V2 players can play LASeR V1 content like LASeR V1 players. The LASeR binary encoding is generic and extensible, it is also backward and forward compatible.	Yes	Backwards compatible as MORE is based on open standards and does not alter the semantics of existing components. For e.g. content that is based on SVGT 1.1 can be fully rendered on SVG Mobile 1.2 (MORE) user agent.
	RME-IOP-002 Old versions of the RM enabler SHALL be forward compatible	Yes	LASeR inherits from SVG Tiny a provision for versioning so that LASeR V1 players can play LASeR V2 content to the extent possible The LASeR binary encoding is also backward and forward compatible.	Yes	The design of MORE based on open standards and therefore is extendable.

Caching Storage requirements	RME-FUNC-018 The storage of RM data and the privacy to be applied to the stored information SHALL be possible on the client and/or on the server side	Yes	The use of LASeR does not impact the storage and management of the privacy of data on the server. LASeR allows to store and manage privacy of data on the client by providing an interface to store information on a device, with cookies-like limitations for security.	No	MORE utilizes existing mechanisms and it is important to note that MORE does not preclude using a particular method available on the terminal. At this time browser environment caching is supported by most implementations, either through proprietary means or through client/server standard mechanisms such as cookies.
	RME-FUNC-019 RM data and update SHALL be cachable locally i.e.: on the end user device.	Yes	SAF provides means to specify the cacheability of streams and scenes on the client device. LASeR provides an interface to store information on a device	No	MORE utilizes cache mechanisms of the parent application (e.g. Browser). It is important to note that caching is application specific functionality and can be different based on the application in use. We recommend that OMA define this functionality that is suitable to RM enabler. At this time browser environment caching is supported by most implementations, either through proprietary means or through client/server standard mechanisms.
	RME-FUNC-020 The RM enabler SHOULD be able to manage preference data (user and/or application) locally.	Yes	LASeR provides a way to manage preference data locally through the use of LASeR Commands Save, Restore and Clean.	No	MORE utilizes existing mechanisms and it is important to note that MORE does not preclude using a particular method. It is not yet clear whether an RME specific personalization and caching mechanisms will be required, or whether RME will be able to use an externally defined personalization or

				caching enabler. At this time browser environment personalization and caching is supported by most implementations, either through proprietary means or through client/server standard mechanisms such as cookies.
RME-FUNC-021 RME-SEC-001 The RM enabler SHALL NOT allow to share private data from one service to an other (e.g.: allocation of data to a dedicated service based on cookies-like functionality)	Yes	LASeR uses a "cookies" mechanism which provides exactly this	No	MORE utilizes existing mechanisms and it is important to note that MORE does not preclude using a particular method. At this time browser environment personalization and caching is supported by most implementations, either through proprietary means or through client/server standard mechanisms such as cookies.
RME-FUNC-022 End-User privacy SHALL be respected	Yes	LASeR uses a "cookies" mechanism which provides exactly this.	No	MORE utilizes existing mechanisms and it is important to note that MORE does not preclude using a particular method.
RME-SEC-001.1 The RM Enabler SHALL be able to securely store permanently a small amount of information for personal information purposes and RM session contexts (i.e., stateful session, icons,,user preferences…)	Yes	LASeR and SAF use a "cookies" mechanism which provides exactly this .	No	MORE utilizes existing mechanisms and it is important to note that MORE does not preclude using a particular method. It is not yet clear whether an RME specific personalization and caching mechanisms will be required, or whether RME will be able to use an externally defined personalization or caching enabler. At this time browser

					environment personalization and caching is supported by most implementations, either through proprietary means or through client/server standard mechanisms such as cookies.
	RME-SEC-001.2 The RM Enabler SHOULD be able to securily store temporary a large amount of persistent information for content cache process and offline navigation.	Yes	SAF provides suitable caching hints to achieve such functionality on a best-effort basis (i.e. if memory is available on the device to achieve the caching). LASeR provide an interface to store information on the device See Section 6.6.2.1 of the LASeR specification	No	MORE utilizes existing private storage mechanisms of the parent application (e.g. Browser) as applicable.
	RME-SEC-001.3 The RM Service SHOULD be able to define content storing mechanisms and the storing priority according to the rich-media service logic.	Yes	SAF provides suitable caching hints to achieve such functionality on a best-effort basis (i.e. if memory is available on the device to achieve the caching). LASeR provide an interface to store information on the device	Yes	MORE utilizes standard caching mechanisms for HTTP.
Requirements for the packaging format	RME-FUNC-004 It SHALL be possible for the service provider to aggregate RM data	Yes	SAF answers to this requirement. Key benefits using SAF are: 1 - interleaving of media · 2 - Precise Synchronisation mechanism 3 - Decrease round trip and network delay 4 - Decrease network request 5 - Low file overlay	Proposed (3GPP)	MORE uses the ISO Base Media File Format to aggregate media (SVG, audio, video, raster and vector graphics) in to a single delivery/container format. We define a new media box for SVG and provide information for SVG to interact with existing media (e.g. audio, video) present in the container format. The timing synchronization provides the interfacing with these multiple media, and do not see a need to add a new stream to the file format.

	RME-FUNC-005 It SHOULD be possible for the service provider to aggregate RM updates	Yes	Same as above. In addition SAF allows the integration of additional updates or streams in a file or stream whose transmission has already started.	Proposed (3GPP)	The ISO Base Media File Format can aggregate media (SVG, audio, video, raster and vector graphics).in scene updates. Please refer to the container format section in the MORE proposal.
Requirements for the Transport mechanisms	RME-FUNC-010 Efficient transmission (low delay, low overhead) of RM data and updates SHALL be provided	Yes	SAF is bit efficient and provide a very low overhead on data.	Proposed (3GPP)	MORE supports efficient transmission including effective packetization and fragmentation. Fragmentation is needed when an entire sample cannot fit in one transport packet. The packet size depends on several factors such as the server's capability, operator, network conditions, etc. Packetization involves packetizing samples an/or one or more of their fragments into transport packets. Packetization is provided via RTP payload packet formats defined in the MORE proposal.
	RME-FUNC-011 RM content SHALL be available in streaming, progressive download and download.	Yes	LASeR inherits the progressive download capability from SVGT1.2 and extends it to provide complete and well-formed fragments/packets for checking and optimisations purposes. The LASeR design applies to the scene and rich-media data the conception of streamability that apply today to AV. A payload format to map LASeR over RTP is also defined. The aggregation format SAF can also be packetised over RTP using the same payload format.	Yes Proposed (3GPP)	Download and progressive download are already possible using SVGT 1.2 In addition, the ISO Base Media container format also provides ability for download, progressive download and streaming profiles. MORE provides packetization of SVG for streaming purposes. The ISO Base Media Files are used by the streaming server to obtain synchronization and hint track information to form RTP packets for

					streaming
	RME-REL-001.4 The RM service SHOULD be able to accept content with a range of packet size limits, as defined by the content provider.	Yes	The service provider can define the packet size of LASeR content in order to fit to the networks limits and to decrease the network latency.	Yes	MORE provides for fragmentation of data based on packet size limits as set by the content/service provider.
	RME-NI-001 The RM enabler SHOULD be able to be interface with any and/or multiple bearers simultaneously	Yes	The design of the LASeR engine supports the integration within one scene of data and streams coming from various origins, e.g. different HTTP servers, DVBH + 3G, HTTP + RTP, etc. This can be achieved through the 'href' attribute or through StreamHeader and RemoteStreamHeader units.	Proposed (3GPP/OMA)	MORE supports rich media delivery using either 1-to-1 bearers (for e.g. PSS, HTTP) or 1-to-many bearers (for e.g. MBMS, FLUTE/ALC) or a combination of both. Depending on the service and the terminal capabilities, an appropriate set of bearers can be chosen to deliver the rich media content. Some of these bearers can also be used simultaneously.
	RME-NI-002 The RM enabler SHOULD be able to address bundle of stream	Yes	LASeR is transport-agnostic and can interface with any channel bundling, e.g. MPEG-2 PS/TS, SAF is specified for stream aggregation	Proposed (3GPP)	MORE contains provision for depacketization of streamed media and consequent presentation. Packetization and depacketization are provided based on the RTP payload packet formats defined in the MORE proposal. At the enabler, the depacketizer is used to depacketize the RTP packets.
Integration in the mobile environment requirements	RME-SYS-001 The RM enabler SHOULD be able to interface with other resident clients on the phone.	No	LASeR engines can be interfaced with other applications such as SMS and MMS clients, A/V clients, etc.	No	As MORE does not alter the semantics of the clients' rendering language, it provides for a flexible user agent that can be easily integrated with other enablers in an OMA environment. At the architectural level the content will use the same DOM definition across

				XHTML and SVG allowing document interaction. At the API level, MORE will be able to interact with JAVA applications through the shared DOM and the use of the JSR-226 API At the application level MORE will support the URI schemes mechanism for invoking other applications. In addition, as MORE is based on xml the RM enabler is able to interface (share DOM, have script work over document boundaries, reuse components like xml-parser, etc) with the other xml-based technologies on the device (XHTML, SMIL, CDF, XForms).
RME - System Element A (browser) The RM Enabler SHOULD be able to interface with browser client	Proposed Being specified	LASeR engines can be packaged as plugins to existing browsers. LaSeR v1 does not specify specific interface for the browser, and let the usage of uDOM to the implementation phase. LASeR v2 specify an extended uDOM as an interface to the browser.	Yes	As MORE does not alter the semantics of the clients' rendering language and is xml-based, it provides for a flexible user agent that can be easily integrated with other enablers in an OMA environment. At the architectural level the content will use the same DOM definition across XHTML and SVG allowing document interaction. At the API level, MORE will be able to interact with JAVA applications through the shared DOM and the use of the JSR-226 API It is very important to note that although MORE is based on SVG Mobile 1.2 it can be easily extended to

				support CDF documents in future.
RME-SEA-001 The RM Enabler SHALL be able to launch the browser	Yes	LASeR engines can launch external applications, including the browser.	Yes	MORE has the provision for allowing RM applications to launch the browser by using SVG 1.2 mobile profile's linking and scripting capabilities.
RME-SEA-002 The RM Enabler SHALL be able to be launched by the browser.	Proposed (OMA/3 GPP)	LAseR engines can be launched by the browser. Nothing in the LASeR specification precludes it and existing implementations have already implemented this feature. A mime type is to be specified	Yes	MORE has the provision for allowing the browser to launch RM applications using existing plug-in architecture.
RME-SEA-003 The RM Enabler SHOULD be integrated as a plugin into a browser	Proposed (OMA/3 GPP)	LASeR engines can be plugins to browsers. A mime type is to be specified	Yes	MORE has the provision for integrating RM applications into the browser by using existing plug-in architecture.
RME-SEA-004 The RM Enabler SHALL expose the uDOM API to the browser	Proposed (OMA) Being specified	LASeR V1specification leaves to the implementation the integration of UDOM within the browser. MAE will need to decide either to leave the usage of uDOM to the choice of implementers or to mandate the usage of uDOM along with LASeR v1 LASeR V2 specification will integrate the uDOM and extend it to the LASeR scene tree extensions. The LASeR dynamic update mechanism can be implemented easily above uDOM. An informative mapping is provided in the LASeR specification.	Yes	MORE is based on SVG Mobile 1.2 profile, and is therefore designed to expose the uDOM API to the browser.
RME-SEA-005 The RM Enabler MAY provide other API to the browser	No	Other API to the browser can be defined above the LASeR user agent. Care has been taken to allow the inclusion of LASeR engines into CDF-compliant applications.	No	MORE is extendible and may provide other API to the browser if compatible with the SVG Mobile 1.2 specification.
RME - System Element B (AV codec) The RM Enabler SHALL be able to address and to provide a tight integration with AV codec.	Yes	LASeR is designed with a tight integration into the MPEG terminal model in mind, which allows efficient interfacing with any kind of media. Sections 3.1.1, 6.3, 6.4 and 7 define the connection of LASeR with the MPEG terminal model. LASeR additionally defines overlay = "fullscreen" to improve the usability of videos on mobile devices.	Yes	The MORE UA has tight timing synchronization and architectural integration with the AV codecs associated with SVG Mobile 1.2 profile. MORE utilizes the run-time synchronization functionality that SVG Mobile 1.2 inherits from SMIL. These

				attributes are syncBehavior, syncTolerance and syncMaster attributes, specified on the 'audio', 'video' and 'animation' elements, and syncBehaviorDefault and syncToleranceDefault attributes specified on the svg element.
RME-SEB-002 The RM Enabler SHOULD be able to access Metadata stream.	Yes	LASeR engines can interface with any kind of streams. LASeR provides the same features as SVG for this. SAF can carry any type of stream, and LASeR can refer to any type of stream.	Proposal (3GPP)	MORE supports metadata information such as media description, session description, SVG scene similarity, etc. are provided. This metadata information is stored in the container format, is used for forming RTP packet types for the purpose of streaming.
RME-SYS-002 The RM enabler capabilities SHALL be expressable within UAPROF	No	LASeR engines capabilities are easy to express with UAprof. This should be achieve by using other OMA's enablers	Proposed (OMA)	MORE will utilize existing UAPROF capabilities of the parent application (e.g. Browser), as well as the capabilities being developed as part of OMA's Device Profiles Evolution (DPE) work.
RME-SYS-002.1 The RM enabler capabilities SHALL be advertisable by the browser or by the rich-media enabler depending on the usage scenario	Proposed (OMA/3 GPP)	LASeR engines can advertise their capabilities in a variety of ways, including the use of HTTP Accept Headers and Media Queries. We propose to use this to method for that. In complement other OMA's enablers can be used	Proposed (OMA)	MORE will achieve this via OMA's UAProf and DPE solutions.
RME-SYS-002.2 The RM enabler and the rich-media service SHOULD benefit from underlying support of dynamic UAPROF service.	No	LASeR engines can be interfaced with underlying support for dynamic UAprof services. This should be achieve by interfacing with other OMA's enablers	Proposed (OMA)	MORE will integrate the OMA's DPE work which addresses the dynamic profile update capability.
RME-FUNC-023 The service provider	No	LASeR engines can be interfaced with underlying support for dynamic UAprof services.	No	MORE can use protection mechanisms (e.g. DRM) that exist

SHOULD be able to protect the RM content.		This should be achieve by interfacing with other OMA's enablers		for media constituting the RM content.
RME-SEC-002 The RM enabler SHOULD be able to interface with DRM client	No	LASeR engines can be easily interfaced with DRM tools in order to provide appropriate protection to RM content.	No	MORE can be interfaced with DRM mechanisms that exist for media constituting the RM content.
RME-IOP-003 Service enabled by the RM Enabler SHALL be available whilst the user is roaming on a different network which is capable of RME services.	Yes	This is orthogonal to any media type. No special roaming services are required to allow mpeg' part 20 data to be used across any bearers and networks.	Yes	Services provided by MORE is agnostic of any network as long as the network conforms to existing standards based features to support such services. MORE uses transport protocols such as RTP, FLUTE/ALC, and HTTP based on existing standards. No special roaming services are required to allow MORE data to be used across networks.

Table 2: RME Requirements Table

Table for DIMS requirements, relevant for RME.

Requirement Groups and detailed requirements		MPEG 4 part 20		Other technology #1	
		Supports	Comments	Supports	Comments
DIMS complementary general requirements for the mediatype	Ref: S4-050800 section 4.1 Number 2	Yes	LASeR inherits its rendering model from SVG Tiny 1.2, thus achieving compatibility by equality.	Yes	MORE is centered on SVG Mobile 1.2 specification. Therefore it is fully compatible to rendering models of the SVG Mobile 1.2 specification.
	Ref: S4-050800 section 4.1 Number 7	Yes	This requirement was also an MPEG requirement when starting the LASeR works.	Yes	MORE relies on existing highly efficient mechanisms for compressing embedded media such as audio, video and raster images. For SVG, MORE uses GZIP that is proven to offer high compression rates for large graphics content. For small graphics content, MORE encourages the use of raw XML data due unnecessary encoding/decoding modules, and as a result extra footprint. Note: The choice of compression format is not

					dependent on the comparison between content or packet size.
	Ref: S4-050800 section 4.1 Number 10 a	Yes	SVG-style embedding of images and clips inside the scene description is possible, but not recommended since it is not an efficient use of the MPEG terminal model. If used, this feature does not incur the typical Base64 encoding overhead that SVG incurs. SAF packaging does not impact the compression efficiency beyond a very small fixed overhead per packet (as opposed to Base 64 encoding in multipart packaging).	Proposal (3GPP)	A choice for a container format in MORE is the use of the ISO Base Media File Format. This format allows the packaging of SVG scenes and scene updates along with other media. Information such as timing synchronization, hint tracks, random access are provided. As these media can be stored in this container format with their own compression mechanisms, the format does not interfere or reduce compression efficiency of the media.
	Ref: S4-050800 section 4.1 Number 14	Yes	LASeR does not mandate the usage of a particular font system and then allows to use any font solution (native, device capability, SVG font, Opentype font, other) A fallback mechanism is provided as per SVG.	Yes	MORE fully supports the notion of using terminal-supplied (or system) font capability and this is inline with the requirement for existing mobile SVG implementations.
	Ref: S4-050800 section 4.1 Number 15	Yes	LASeR is extensible in many point of view: 1 – the scene description is extensible 2 – The dynamic update mechanism is extensible 3 -The binary encoding is extensible (other XML data can be encoded and transmitted e.g.: proprietary data, CDF)	Yes	The primary motivation behind the architecture of the MORE system is the need for a strong separation of interfaces and layers. By enforcing such a strong separation, allows us to pick a best of breed approach, and to change it over time if necessary.
	Ref: S4-050800 section 4.1 Number 22	No Being Specified	LASeR v1 doesn't provide any specification for that. LASeR v2 introduces media chain stats information through a new set of XML events.	Yes	MORE uses "preload", "postload", and "loadProgress" events to realize the state of media chains and make available at the scene level. In addition, the application can make use of the various attributes defined by the ProgressEvent interface to understand the details of these event types.

	Ref: S4-050800 section 4.1 Number 23	Yes	Rectangular clipping is supported in LASeR and allows display of rectangular parts of any content and it is particularly suitable for images as well as for text display management.	Proposed (W3C SVG Full)	MORE utilizes the relevant features such as ref() transform value, clip preserveAspectRatio attributes from SVG specification. Please note the clip attribute is not currently supported in SVG Mobile 1.2 specification but only in SVG Full due to implementation feedback based on complexity issues that arise from mandating this feature. However, MORE does not prevent use this feature if it is required to do so and the implementation support for this feature is provided.
DIMS complementary updates and interaction requirements for the media type	Ref: S4-050800 section 4.2 Number 7	Proposed (OMA) Being Specified	LASeR commands can be used in parallel with a scripting language using a uDOM interface. As the LASeR scene tree is an SVG Tiny 1.2 scene tree, a uDOM interface can be implemented within a LASeR v1 client. LASeR v2 will integrate and extend the uDOM to the LASeR scene extensions.	Yes	MORE re-uses the scripting functionality and bindings as defined by SVG Mobile 1.2 specification and JSR 226, and therefore supports both scripting languages (Ecma and Java).
	Ref: S4-050800 section 4.2 Number 10	Yes Being specified	The LASeR v1 specification defines an informative mapping of the LASeR Commands to uDOM instructions using ECMA-Script, thus proving their implementability with uDOM. LASeR v2 will incorporate the uDOM within the specification	Being Specified	MORE relies on the REX (Remote Events for XML) initiative, a joint effort between W3C SVG and Web Apps working groups. The proposed XML update specification will be based on a set of requirements that are intended to maintain compatibility with DOM events, declarative in nature, and integrates well with the WWW architecture. REX encoded messages can easily implemented using the uDOM API using methods such as appendChild(), removeChild(), and setXXXTrait().
	Ref: S4-050800 section 4.2	Yes	As a LASeR scene tree is an SVG Tiny 1.2 scene tree, LASeR Commands apply equally to	Being Specified	Same as above

	Number 11		LASeR and to SVG Tiny 1.2.		
	Ref: S4-050800 section 4.2 Number 12	Yes	The DOM Level events are part of the LASeR specification	Yes	MORE preserves the DOM Core and DOM Level 3 events model and therefore remains fully compatible with local interactivity as defined by these models.
DIMS complementary requirement for Application features	Ref: S4-050800 Section 9 Number 3	Being SPecified	The LASeR v2 provides a stream source mechanism for that	No	MORE believes this is an implementation dependent feature and application specific.
DIMS complementary Caching / Storage requirements	Ref: S4-050800 Section 9.1 Number 6	Yes	The LASeR specification provides a Cache unit mechanism for this functionality. See LASeR specification, clause 7.9	Yes	MORE follows the SVG Mobile 1.2 display and animation modules among other to enable pre-loaded content.
	Ref: S4-050800 Section 9.1 Number 7	Yes	The LASeR specification provides a Cache unit mechanism for this functionality. See LASeR specification, clause 7.	No	MORE treats this as an application specific feature.

Table 3: DIMS Requirements Table

7 Evaluation of the technologies against the requirements

In this section the technologies are evaluated against the following criteria

- Ability to meet the RME requirements

7.1 MPEG4 part 20 (LASeR)

This section provides an evaluation of an MPEG4 part 20 based solution against the RME requirements and includes additional technologies to meet the need of a complete Rich-Media Enabler

7.1.1 Alignment of LASeR with SVG Tiny 1.1 and 1.2

There is a clear consensus that the RME/DIMS enabler will be based on the SVG Tiny 1.2 specification.

LASeR is an MPEG encoding of the W3C SVG Tiny specification, then does not mandate nor prevent the XML parser and the gzip compression to be used as defined in the SVGT1.2 specification. Compliancy with the rendering model is provided as described in the figure below:

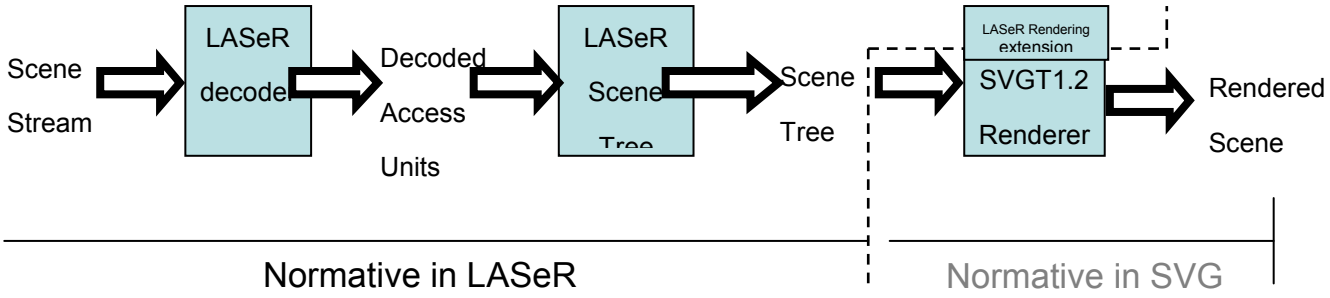


Figure 1: LASeR engine components and normative parts (from section 6.4 of the LASeR specification)

LASeR v1 extends the feature set of SVGT1.1, including features of SVG1.1 Full and SMIL2 which will be present in SVGT1.2.

LASeR first amendment (called LASeR v2 in this document) will be a superset of SVGT1.2 and will complete the alignment with the not-yet-stable features of SVGT1.2.

LASeR v1 is already able to encode and transmit SVGT1.2 (and other XML data, e.g.: proprietary extension, CDF) content due to its generic, extensible binary encoding scheme.

The non-v1 part of the SVGTiny 1.2 or LASeR v2 content will be skipped by a LASeR v1 decoder, will be rendered by a LASeR v2 decoder. It can also be transmitted to an SVGTiny 1.2 player depending on the implementation choice specify by OMA.

In this document when LASeR is mentioned, it refers to features that are relevant in LASeR v1 and v2. When features are only relevant for one version, it will be explicitly mentioned.

7.1.1.1 LASeR scene extensions

The LASeR scene extensions cover:

- The management of any input device to ease the content adaptation to any particular MMI and terminal.
- The association of a precise timing model to any attribute.
- The clipping by a pixel-aligned rectangle with horizontal and vertical borders, which is crucial to create UI widgets.
- The possible use of any font system, including OpenType.
- A fullscreen mode for videos and images.
- A means to stop non-rendered animations to optimize CPU usage.
- The use of the SMIL mediaClipping module to allow VCR-like control of media.
- A simple way to underline text.

The overview of the components of a LASeR client and of the global architecture using MPEG4 part 20 for an application is as follows:

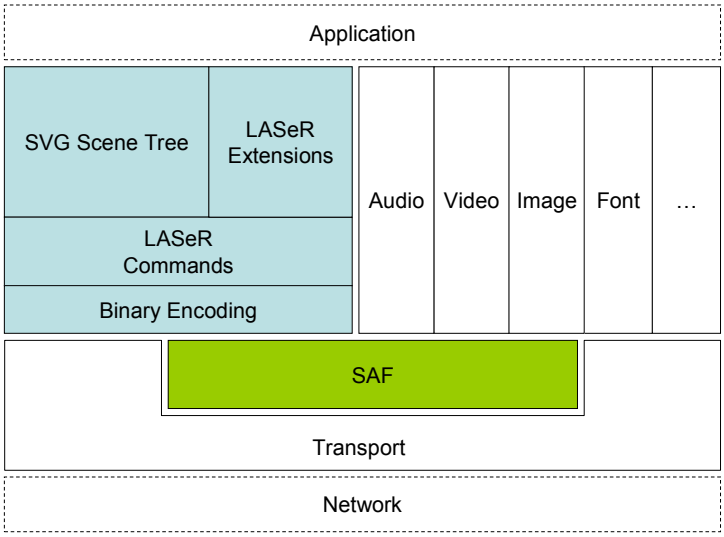


Figure 2: Architecture of LASeR and SAF

7.1.1.2 Font

LASeR does not mandate any font system but recommends the usage of Open Type fonts. So the preferred way of sending font information with LASeR is as a companion font stream using the OpenType format. However, there are other options: SVG Tiny fonts can be sent separately in a companion stream (encoded as OpenType or in XML form), or in the LASeR scene, encoded with the anyXML encoding (non-schema encoding of any XML data). An implementation of a LASeR client with a SVG font engine would be able to render the SVG font included in the SVG/LASeR content.

It is also possible to translate SVG Tiny fonts to OpenType format.

When a requested font is not present, LASeR provides the same fallback to system/device fonts as SVG Tiny 1.2.

7.1.2 RME Dynamic updates requirements

One key additional feature provided by LASeR over the SVG Tiny 1.2 specification is the ability for dynamic modification of the scene.

Dynamic updates are a key to efficient representation of server-driven or user-triggered scene changes over time. This feature, present in Macromedia Flash, is necessary to enable:

- The efficient representation of streamable cartoons,
- The partitioning of scenes into small packets that fit in size-limited delivery mechanisms (such as cell broadcast),
- The dynamic creation of answers to a user request, and their integration in the current scene,
- Or the dynamic push of content into an existing scene.

The dynamic update mechanism can be achieved with two complementary technologies: using LASeR Commands and using a scripting mechanism.

7.1.2.1 Using the LASeR command

The LASeR Commands are a declarative way (as opposed to programmatic as in a script) of specifying changes to the scene. The following commands are defined:

7.1.2.1.1 General commands

- ***Insert***: to insert any element in a group, a point in a sequence.
- ***Delete***: to delete any element by id or from a group by index, a point in a sequence.
- ***Replace***: to replace an element by another element (by id or from a group by index), or to replace the value of any attribute of any element.

7.1.2.1.2 Commands specified for streaming and broadcast

- ***NewScene***: to create a new scene.
- ***RefreshScene***: to repeat the current state of the scene, for use as a random access point into the LASeR stream or as a means to recover from packet loss.

7.1.2.1.3 Commands defined in LASeR for additional requirements

- ***Add***: similar to replace, but with the notion of adding to the value rather than replacing it.
- ***Save***, ***Restore*** and ***Clean***: to save, reload or remove persistent scene information in the form of the value of a list of attributes. Other commands have no influence on persistent scene information.
- ***SendEvent***: to send an event to any element in the scene.

7.1.2.1.4 Extensibility and genericity

LASeR includes a mechanism to extend the LASeR Commands to add other functionality.

LASeR Commands are not specific to LASeR, but can be used on any XML document with minimal extensions. ISO/IEC 15938-1 defines a similar mechanism as the group of commands in 7.1.2.1.1, with slightly different requirements, proving the applicability of the concept to any XML document. One possible application of XML document update commands generalized from LASeR Commands is to Compound Document Format (CDF), and more specifically to Web Interactive Compound Documents (WICD) which are based on a mix of xHTML and SVG Tiny 1.2.

The LASeR specification defines an XML syntax (LAseRML) for use in authoring or other applications of XML versions of the LASeR scenes. LASeRML is a superset of the SVG Tiny (XML) syntax. LASeR Commands, as part of the LASeR specification, also have an equivalent XML syntax, which is immediately applicable to SVG Tiny 1.1 and 1.2 documents.

7.1.2.1.5 Timing model

A timing model is associated to the LASeR commands, allowing the player to provide a very tight synchronization, with an accuracy specified by the content creator wishes (frame accurate synchronization, synchronization on a user interaction, on a time basis, etc…). This timing model defines the link between the time stamps used by transport layers and the scene time or

composition time and is the key to any streaming and/or synchronization of scene information with other media. See § 6.4 in the ISO/IEC 14496-20:2006 specification

7.1.2.1.6 Compatibility Issues

LASeR scenes and updates are defined as complete and well-formed packet. The first LASeR packet contains a complete, well-formed SVG Tiny scene (with end tag) which represents the first state of the content. The next LASeR packets are sets of commands (with end tag) to build the next states of the content. After each packet is received and each update command is executed, the scene in the browser is a valid, well-formed SVG scene.

In the LASeR v1 specification, an informative ECMA-Script/DOM equivalent of the LASeR Commands is provided. Using this equivalent code, LASeR Commands can be implemented at minimal cost on SVG Tiny 1.2 implementations including a DOM interface and an ECMA-Script interpreter. This informative equivalent also serves as an indication of the complexity of the implementation in compiled languages on top of an SVGTiny1.2+DOM player, as the total complexity of the group of commands in 7.1.2.1.1 is less than 100 lines of code.

LASeR extends the feature set of SVG Tiny, and as such, reuses DOM Level 3 Events, also known as the XML Events specification in order to provide a generic extensible

The usage of uDOM in LASeR v1 is possible, but not mandated. LASeR v2 will specify the usage of uDOM and its extensions to the (few) LASeR scene tree extensions.

7.1.2.2 Updates through Scripting

In addition or in parallel to the LASeR command, the use of scripting and DOM Network API and an ad-hoc protocol to communicate scene modification from the server to the client can be used. Note: the extra cost incurred by defining an alternate protocol in script and the requirement of an ad-hoc server makes this solution only worthwhile in very specific services.

7.1.3 Combination of updates

LASeR Commands are used in two contexts:

- in a timed context
- in an interactive context

LASeR Commands are used in a timed context when they are part of a LASeR Access Unit. The LASeR Access Unit has a presentation time which is the time at which the LASeR Commands in it shall be executed. LASeR Commands from a LASeR Access Unit are executed in step 3 of the LASeR execution model (section 6.4 of ISO/IEC 14496-20). Such LASeR Commands can never interfere with scripts with another scriptContentType, since these are executed as part of step 4.

LASeR Commands are also used in a non-timed, interactive context when they are contained in a script element. Upon activation of the script element, e.g. through an event channelled to the script element by a listener element, the LASeR Commands are executed as if their presentation time was the current scene time. LASeR Commands from a script element are executed in step 4 of the LASeR execution model (section 6.4 of ISO/IEC 14496-20). Interaction between the execution of LASeR Commands in a script and the execution of DOM calls by a script with another scriptContentType is resolved by the processing order of the events which trigger the scripts' execution.

DOM calls would apply to a LASeR v1 scene tree in a LASeR v1 implementation that also implements uDOM.

Since the execution of the two flavours of LASeR Commands are clearly specified to happen in different steps of the LASeR execution model, there can be no unforeseen interference between the two. The author can precisely predict what will happen. For two script executions happening within the same rendering cycle, the same rule shall be applied to order any mix of LASeR Command script and script with other scriptContentType.

7.1.4 Streaming and reliability requirements

The LASeR format allows streaming over reliable and non reliable network. As SVG Tiny 1.2 specification LASeR supports the following scenarii:

- The first option is the classical “download and play” mode. The user waits until the end of the download to start viewing the content.
- The second option is the progressive rendering mode. This mode is an improved version of the previous one enabling visualization while downloading the content. But the downloaded content only adds new content to the existing one, making it difficult to manage long-running documents.

In addition to this, LASeR supports true streaming, allowing long-running documents with a high-rate of updates, such as cartoons or vector graphics commercials, as well as the synchronization of streamed scene information with other media.

7.1.4.1 Progressive download and rendering

7.1.4.1.1 In SVGT 1.2

SVGT1.2 introduces progressive rendering, and a mode where the scene time can begin to progress and rendering can start before the end tag is received. Thus, players cannot rely anymore on the reception of the end tag to make integrity checks. If any packet is lost, the SVG decoder will reject the content and stop rendering.

In order to allow rendering before the end of the download, SVG constraints on well-formedness have to be dropped. Once the end tag has been received, nothing else can ever be sent any more, so the end tag is only received when the scene is at end. In the case of an interactive scene, in order to leave to the user the opportunity to interact, the scene needs to be left open, so the end tag is never received. As a result, a streamed SVG scene is never well-formed. The SVGT1.2 specification works around this problem by defining the well-formedness of SVG fragments.

7.1.4.1.2 In LASeR

LASeR scenes can be modeled as a series of SVGTiny scenes. The first frame consist of the initial SVGTiny scene, the next frames contain the differences, i.e. the set of scene updates required to transform the previous scene into the next scene.

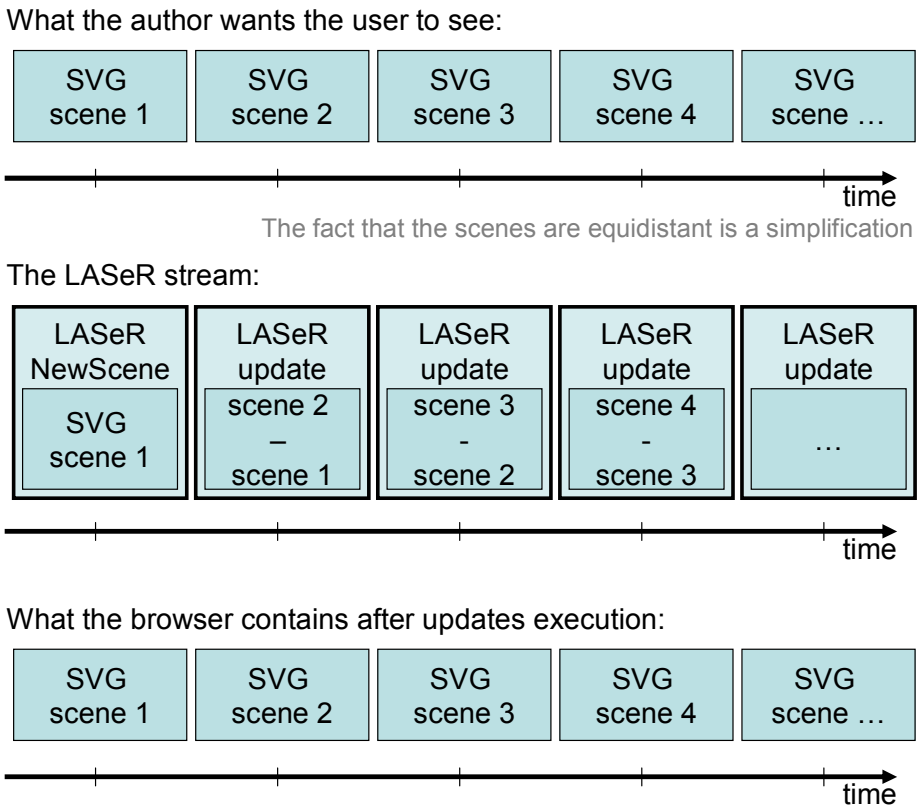


Figure 3: updates construction

The first LASeR packet contains a complete, well-formed SVGTiny scene (with end tag) which represents the first state of the content. The next LASeR packets are sets of commands to build the next states of the content. Each packet is complete and well-formed. After each packet is received and each update command is executed, the scene in the browser is a valid, well-formed SVGT scene.

7.1.4.2 Streaming

7.1.4.2.1 In SVGT1.2

Progressive rendering is not streaming. Let us model the reception of an SVGT1.2 scene as a series of packets. Let us further assume for simplicity that each packet contains a single top element (with children): this is not necessary but simplifies explanations. Each packet/top element is received at a certain time, which depends on the network, and is executed ASAP. This is impossible to synchronize, because there is no way to associate a time stamp with a scene time. If the packet is conveyed in RTP, there is no way to translate the RTP time stamp information into scene time, in order to possibly wait before the insertion of the element in the packet. From the other end, the author has no means to specify: this element shall be inserted in the scene at time T.

7.1.4.2.2 In LASeR

LASeR content is always a stream. LASeR introduces the scene updates mechanism, in order to transpose to scenes the well-known structure of video streams: intra-coded frames followed by predictive-coded frames.

In a LASeR stream, the first packet contains the initial (SVGTiny) scene. As a result, at the end of the first packet, an end tag is received, allowing well-formedness checking and other optimizations.

The next packets contain update instructions. The instructions themselves can be expressed in XML or binary, but in both cases are well-formed and complete. The result of the execution of the update instructions is a complete and well-formed SVGTiny scene.

Each LASeR packet has a specific time stamp. This time stamp may need to be adapted to the underlying transport, but the LASeR specification defines precisely how to recover the scene time information from the transport time stamp. The author needs to specify the scene time at which each update will be executed. As a result, precise synchronization of scene updates with media is feasible.

Within the browser, between packets, the content is complete, well-formed SVG content.

LASeR stream can be packetised over RTP using the RFC 3640 payload, other packetisations can be considered.

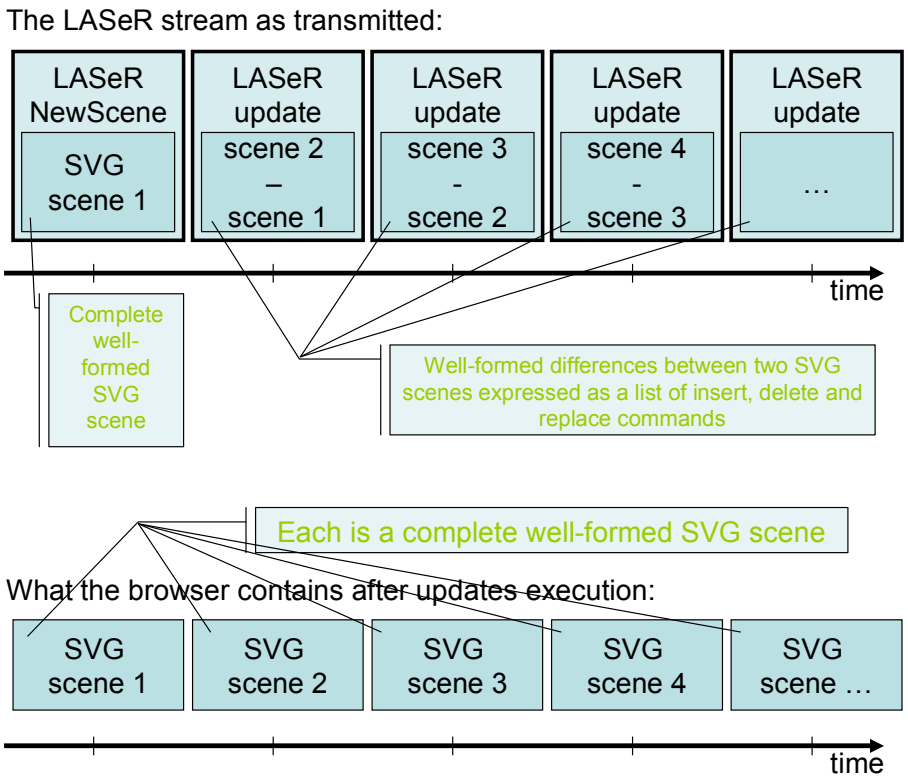


Figure 4: Overview of a LASeR stream

Tuning in into the middle of a scene stream is possible through the use of RefreshScene commands. RefreshScene commands contain a copy of the current state of the scene which can be skipped by all LASeR players but the ones currently trying to tune in. Not all LASeR streams have to contain RefreshScene commands, as many delivery scenarios do not require error recovery (for example, TCP/IP uses packet retransmission to ensure error-free delivery). It is the content author's or provider's choice to include RefreshScene commands into the scene stream.

RefreshScene prove useful both in streaming and in broadcast scenarios.

7.1.4.3 Reliability

The use of non reliable delivery mechanisms (such as RTP) implies potential packet loss. In order to provide error-resilient playerq to be implemented for streamed application, LASeR specifies how to:

- Handle packet loss gracefully: after a packet loss, LASeR commands which have become meaningless are ignored.
  - errors located in packets containing transient information can be recovered naturally
  - errors which cause more significant damage to the scene will cause a refresh request by the user.
- Recover from packet loss:
  - through the use of RefreshScene commands, a player after a packet loss is in a state similar to the "tune in" state.
  - RefreshScene commands are ignored by the players as redundant.

7.1.5 Caching and private data management

LASeR specifies means to achieve data management on both client and server sides. This is achieved partially by the scene format and partially by the packaging format.

In the scene format, LASeR and SAF specify interfaces to:

- local caching of RM data on the end-user device and updating of cached RM data,
- secure temporary storage of a large amount of persistent information for content cache and offline navigation,
- content storing mechanisms and storing priority according to the rich-media service logic,
- private data permanent storage in a memory area reserved by the RM enabler.

In order to protect end-user data privacy, LASeR specifies a cookies-like mechanism to limit the above functionality. LASeR uses signaling similar to the one defined in RFC 2965, which defines a state management mechanism for Rich Media presentations.

7.1.6 Synchronization

LASeR extends the SVG/SMIL timing model, to make it compatible with the MPEG timing model and thus optimize its interfaces with MPEG media decoders.

In addition, together with SAF, LASeR offers a platform for efficient and frame-accurate synchronization of media and scene: both SVG-like scenes with SMIL animations and Flash-like scene with sequences of frames can be synchronized with the best achievable precision.

The next table summarizes the respective synchronization features of LASeR and SVG Tiny 1.2. Note: both require the support from an adequate transport layer to synchronize, such as SAF.

Feature	SVGT1.2	LASeR
Specification of the synchronization of streams	Yes	Yes
Ability to synchronize of events and static animations based on scene time with other media	Yes	Yes
Ability to synchronize scene modifications with other media	No	Yes

Table 4: SVG/LASeR Synchronisation features table

7.1.7 Efficiency

One of the key underlying requirement when designing LASeR was the global efficiency that need to be provided. To fulfill this objective, LASeR provides:

- the dynamic update mechanism,
- an efficient data caching management,
- a binary format, necessary for a fast parsing and a fast, bit-efficient transmission of data,
- an append mode providing means to create fluid, dynamic services, free of the one-new-page-per-request client/server paradigm, as well as making it possible to prepare in advance multiple possible responses to user requests.
- and together with the SAF aggregation format, a means to reduce the number of necessary http connections and the round trip delay.

7.1.7.1 Binary Format

The binary format specified in LASeR allows the encoding of SVG Tiny content. It uses a compact representation for the structure of the SVG elements and uses specific coding algorithms to encode the attribute values of the SVG elements. Because the mobile platforms usually lack hardware float processing, the compression of these attribute values has to be simpler than on other target platforms (PC). Complex computations that would improve the compression ratio by a small amount at the cost of doubling the decoding time have been rejected during the standardization process. Thus, the binary encoding of LASeR is straightforward, and its quality resides in the complexity/efficiency balance. Special care was taken for the encoding of values for some attribute types, like list of float coordinates, vector graphics paths or transformation matrices.

The LASeR binary syntax is extensible, so that private extensions can be mixed among normal LASeR elements and attributes, to be ignored by decoders that do not know how to process them. One possible extension is the encoding of CDF documents with LASeR, which allows the encoding of xHTML and other XML components in the fast-to-parse any-XML encoding extension of LASeR.

As with SVG, small media such as images and short A/V clips can be packaged with the scene. The following should be noted:

- such embedding usually incurs, in SVG Tiny, the 33% compression efficiency penalty inherent to Base64 encoding required for the embedding,
- the same embedding is done in LASeR at no extra cost in compression efficiency,
- as such usage does not follow the MPEG terminal model, it is recommended to avoid this mechanism in favor of the more powerful SAF mechanism.

7.1.7.2 Server side efficiency: the append mode

Many Rich Media services rely on a key feature of LASeR: incremental scenes, made possible by the LASeR append mode. The append mode is the possibility to create a LASeR stream containing not an independent scene, but an addition to another existing scene.

There are two typical use cases of incremental scenes:

- Streaming style: the scene is designed as a sequence of frames, and there is a continuous stream of updates to change the current frame into the next frame. Bandwidth usage is varying but never drops to 0. The incremental scenes of this kind are usually best transported over streaming protocols like RTP. A typical use case is a cartoon-like animation.
- Interactive style: the scene is interactive and user requests are processed by the server. The response to user request is a change to the existing scene, not a new scene. Such scenario also requires continuous updates to the scene, but the statistics of the transmission are totally different from the previous style: bandwidth is heavily used for a short time after a user request, and then drops to 0 until the next user request. Given the variety of usages of mobiles, the next user request could come a few seconds or a few hours later.

From a server-side point of view, the interactive transmissions can be considered as a series of separate connections, as opposed to the continuous connection of the streaming style. It is typically implemented using separate HTTP connections, since each data burst results from a user request. However, from a LASeR viewer point of view, it is the same scene/service that is modified. Hence the requirement for the server to be capable of signaling an append mode: “this stream does not contain a totally new scene, but an improvement to the scene the viewer is currently processing”.

The append mode also allows the creation in advance of multiple responses to possible user requests. If the service is modeled as a state machine, each transition of the state machine represents a change to the current scene and may be implemented as an append component. Careful authoring and scope management is required, in particular to avoid clashes of id between elements added by different append components. Still, this functionality opens the way to servers caching most of the responses to users, therefore dramatically improving the service’s performance.

7.1.8 Packaging

In MPEG-4 part 20, the Simple Aggregation Format (SAF) is defined with the following features:

- Simple aggregation of any type of stream, file or fragment.
- Dynamic addition of new streams/files after the start of the delivery.
- Media interleaving.
- Precise synchronization mechanisms support.
- Signalling of MPEG and non-MPEG streams.
- Optimized packet headers for bandwidth-limited networks to guarantee a very low overhead.
- Easy mapping to popular streaming formats.
- Enhanced support for progressive download.
- Real time transmission/delivery.
- Cache management capability.
- Extensibility such as adding new packet types or new stream types.

7.1.8.1 SAF Elements

The SAF specification defines the binary representation of a compound data stream composed of different elementary streams such as LASeR, xHTML, CDF, SVG, SMIL, CMF/CMX, video, audio, image, font and metadata. Data from these various elementary streams results in one SAF stream by multiplexing them for simple, efficient and synchronous delivery. A SAF stream is made of SAF Access Units (AU) of the following classes:

- AUs carrying configuration information for the media or DIMS/RME decoder to be initialized.
- AUs carrying configuration information for elementary streams not carried inside this SAF stream. Streams that need to be carried separately include streams which are started interactively, or are delivered through another protocol.
- AUs carrying media or Scene AU.
- AUs carrying an end of stream signal, indicating that no more data will be received in an elementary stream.
- AUs carrying an indication that no more data will be received in this SAF session.
- AUs carrying cache units carrying complete scenes to be pre-loaded into the user's cache to speed up the answering time for future requests.

7.1.8.1.1 SAF Benefits

The main objectives of adding SAF are as follows:

- SAF provides a light mechanism that uses low memory footprint (size of the code).
- SAF provides a light mechanism that uses low run-time memory.
- SAF provides a mechanism that enables the addition of media in real time (i.e., during the dynamic composition/creation of the contents).
- SAF provides a delivery mechanism that ensures minimal latency since the content can be parsed and decoded as soon as it is received.
- SAF provides a delivery mechanism that enables to optimize the response size : the response can be interrupted by the end-user when the progressive delivery is in progress, when the emitter detected the interruption it can stop the addition/encapsulation of media within the response and then reduces the response size according to the end-user interest, while maintaining the continuity of service.

SAF provides a delivery mechanism that enables to reduce the number of response when browsing. Compared to WAP delivery mechanism where the number of requests/responses is equal to (N+1) where N is the number of media included in the page/scene; SAF reduces the number of requests/responses to 1.

In addition, SAF may be used as payload format for streaming (over RTP/RTSP) multimedia presentation aggregating Scene data, Still Pictures, Audio and Video. RFC 3640 may be used as an RTP payload format. The improvements of SAF in this use case are as follows:

- SAF enables to improve the synchronization of media that have been encapsulated in the same SAF streams.
- SAF enables to send over RTP scene description and images (vectors or bitmaps).
- SAF enables to reduce the number of RTP streams by aggregating the media and scene description within the same RTP streams: this should induce a reduction of run-time memory and CPU usage (RTP sockets are CPU and memory demanding).
- SAF enables to add in real-time (i.e., during the delivery and content generation processes) media, graphics elements or scene description modifications according to end-user interactions. The end-user interactions are done using a request/response scheme like WAP/WSP or WEB/HTTP.

7.1.8.1.2 Caching and private data management

The packaging format proposed by SAF provides more features for caching / storing mechanism, based on the MPEG model:

- The cacheUnit allows sending a pair url+scene in advance, such that when that url is requested, there is no need for a request to the server. This content pre-load mechanism can be used to optimize the response time for frequent/predictable user requests.
- Each stream can be declared permanent, which means that if the terminal has enough memory, it should store the stream for a duration specified in the stream header. This allows frequently used streams to be labeled specifically so that the device caching module can give them preference.

7.1.8.1.3 Synchronization

Scene formats require support (at least temporal signalling) from the underlying transport mechanism. When the transport mechanism does not provide that support, as is the case with HTTP, SAF provides the support required by the scene format to create a complete platform for efficient and frame-accurate synchronization of media and scene.

7.1.9 Integration

7.1.9.1 LASeR client

The LASeR client is composed of various independent components

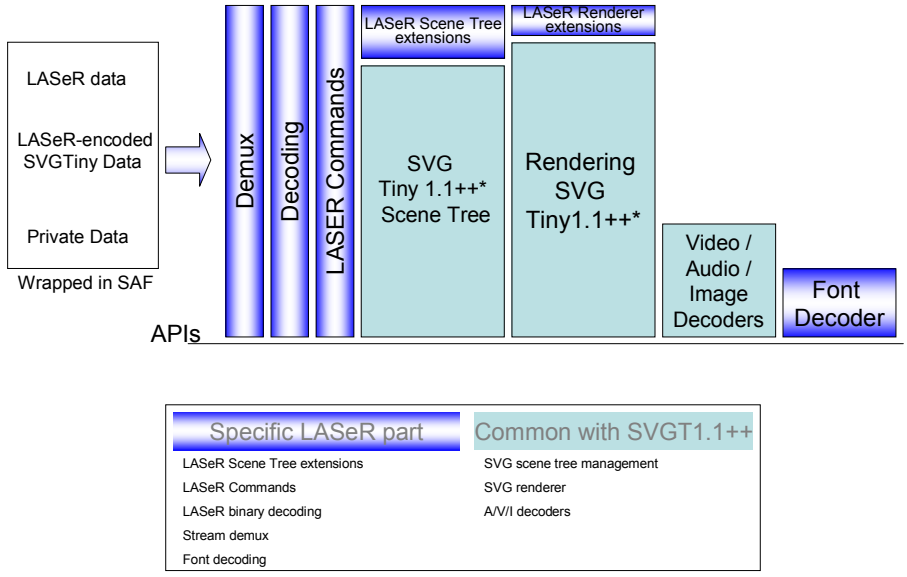


Figure 5: Component architecture of a LASeR v1 client

The Font decoder is not mandated.

*SVGT1.1++ as explain in section 7.1.1

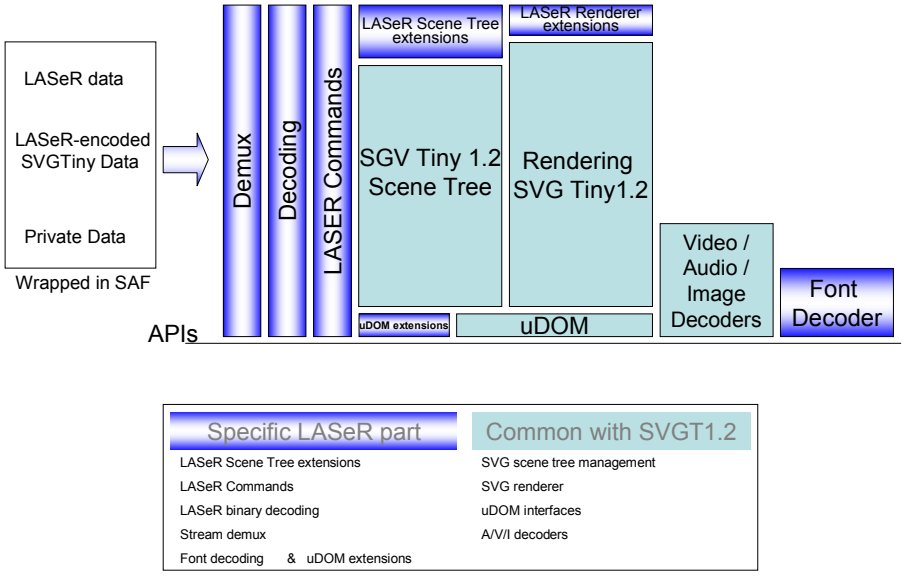


Figure 6: Component architecture of a LASeR v2 client

7.1.9.2 Integration with the SVGT client

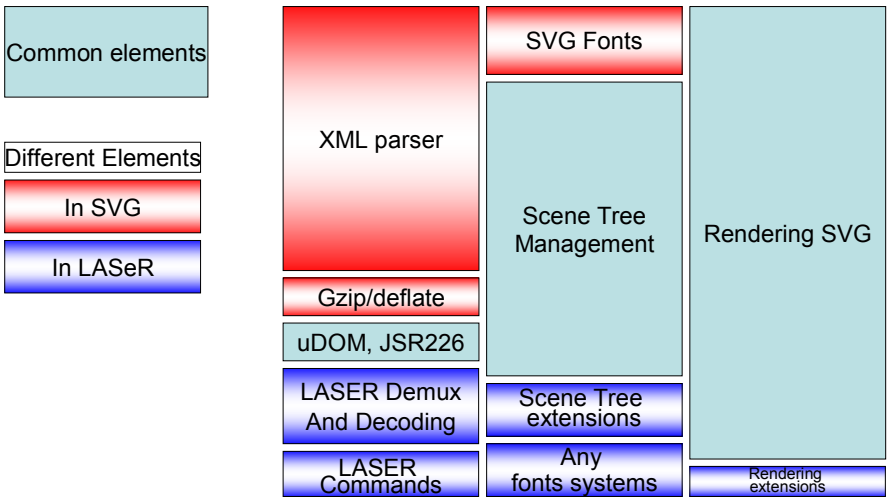


Figure 7: Dual SVG Tiny/LASeR Client

To complement the figure above, the SVG Font subsystem can be a common element.

We estimate a dual player LASeR/SVG Tiny 1.2 to share more than 60% of the code.

The current footprint of the LASeR v1 reference software, (Jar file) in Java, non optimized, is about 100K (excluding SVG Font, codecs, XML parser and uDOM).

7.1.9.3 Integration with the Browser

Same as an SVG Tiny player, the LASeR client or the dual LASeR/SVG client can be integrated in a browser in multiple ways:

- As a plugin: the choice of interfacing is left to the responsibility of implementations, i.e. providing Netscape API or to particular APIs of specific browsers.
- As a plugin using the uDOM API: the integration is more generic and offers interoperable services.
- Integrated according to CDF/WICD recommendations: the integration is generic, offers interoperable services and compound documents are reliably rendered the same way on all implementations.

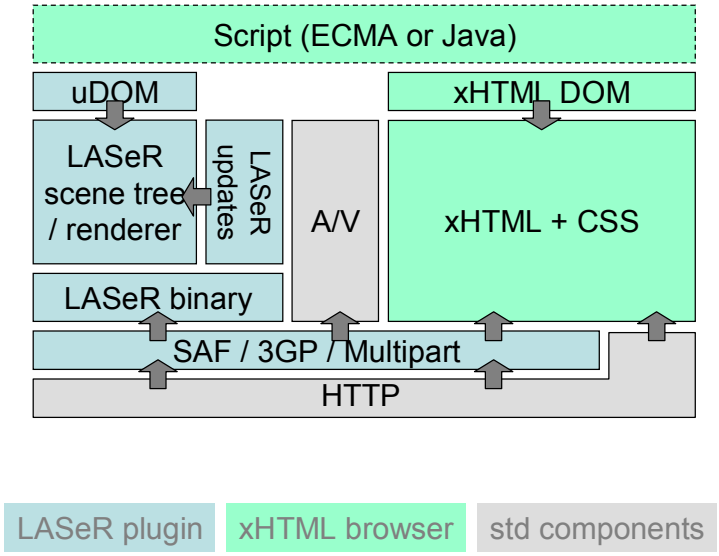


Figure 8: Architecture of LASeR as a plugin in the browser

The above works for a dual LASeR/SVG Tiny 1.2 player as plugin to a browser, and below as a CDF/WICD application.

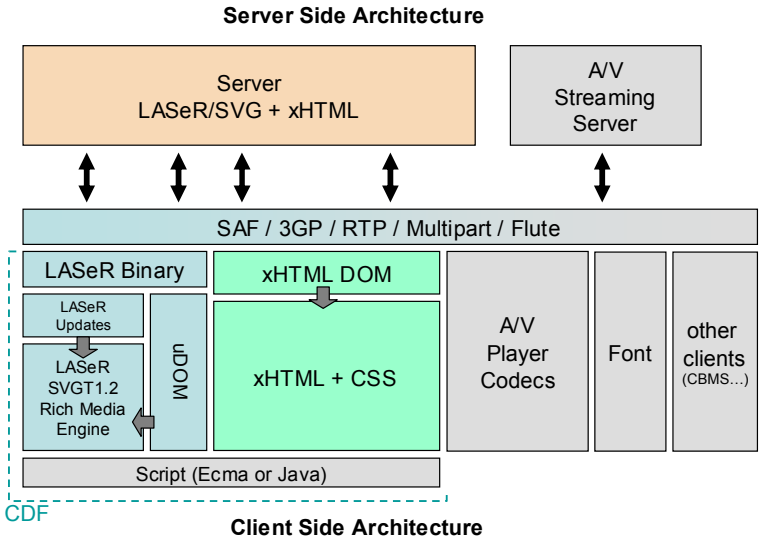


Figure 9: LASeR –CDF Architecture

7.1.9.3.1 Processing Model

Here is a copy of the LASeR execution model :

The playback algorithm of a compliant LASeR Engine shall produce the same result as the algorithm described below with the following high-level steps for each execution cycle:

1. Compute the new scene time Ts (begin of execution cycle);

2. Decode any LASeR AU with a scene time below or equal to Ts, and not yet presented in earlier execution cycles;
3. Execute LASeR Commands from LASeR AUs decoded at step 2;
4. Process all events (DOM, SVG or LASeR) according to the DOM event model [3] and resolve all begin and end times that can be resolved according to the SMIL Timing Model, in clause 10 of [SMIL2];
5. Determine active media objects by inspecting begin and end times,
6. For each active media object, present the media access unit with the normal play time equal to clipBegin + (Ts – begin time) and clamp it using clipEnd.
7. Render the audio and visual element of the scene tree according to the SVG rendering model as described in Clause 3 of [W3C SVG11] (end of execution cycle).

As a consequence the laser processing model does not violate and is compliant with the XML processing model and allows a safe integration within the browser. Some examples are provided in Annex 1.

7.1.9.4 Integration in the OMA environment

7.1.9.4.1 OMA DRM

The specification of the RME enabler is orthogonal to the usage of OMA DRM. Encrypting of AV stream or content as well as decrypting process can be managed by the usual DRM client. Interface can be implemented between the RME enabler and the DRM user agent to provide security message within the rich-media scene. (e.g.: you can not access to this content. To access to this stream it will cost you X€)

DRM consideration are orthogonal to the media-type.

DRM considerations may leads to develop additional specification if protection of the Rich-media content itself or part of it will be required.

7.1.9.4.2 BCAST

The BCAST specifications on ESG and Service allow Rich-Media data to be transmitted and used.

During the interim meeting in Tokyo (September 2005) it has been agreed by the BCAST group that the BCAST specification SHALL NOT preclude the usage of MPEG-4 part 20.

The specificities of LASeR (updates, binary format and streaming) can be beneficial to a BCAST application without modification of the actual BCAST specifications on the following points:

- Bandwidth saving
- Integration of the auxiliary data within a Rich-media service
- Creation of an interactive stream, possibly accessible from the ESG fragment or service.

7.1.9.4.3 Others OMA enabler (DCD…)

LASeR can be interface with other OMA enabler.

In particular, for DCD, specific features of LASeR such as update, streaming can be beneficial both for the creation of a DCD content and for the rendering of a DCD application.

7.1.9.5 Integration in the 3GPP environment

7.1.9.5.1 LASeR in 3GPP delivery

LASeR content can take two forms:

1. LASeR scenes: The first form is like an SVG scene, with only one access unit, and no stream.
2. LASeR streams: The second form is a video-like stream, with multiple access units. Examples include cartoons.

A variant of the first form is a LASeR scene with few access units. This is in principle like a short video clip, and can be assimilated to form 1.

For MBMS, PSS and MMS, LASeR integration is:

- LASeR scenes with just one access unit, or a few access units, behave exactly like gzipped SVG scenes, and should be treated the same way within MBMS, PSS or MMS.
- LASeR streams behave like audio or video streams, and as such, should be treated the same way as video or audio streams within MBMS or PSS. It does not seem appropriate to send large cartoon streams as part of an MMS.
- LASeR uses the payload format RFC3640 for RTP streaming.

7.1.9.5.2 LASeR in 3GPP Extended File Format

LASeR content can be stored within files compatible with the 3GPP Extended File Format. As a LASeR stream is a timed stream, made of AUs, the storage of LASeR streams in 3GP files is straightforward and similar to the storage of audio or video streams. Each LASeR AU is stored as a sample. All these samples form a LASeR track identified by a four character code. The configuration for the LASeR decoder is stored as an entry the sample description box. In case of a LASeR stream comprising only one AU, it is also possible to store this AU, as it is done in the 3GPP specification for SMIL presentation, i.e. as a primary item of the file, using the Metadata box structure.

7.1.9.5.3 SAF and the 3GPP extended file format

SAF is not a file format but a packaging format. 3GPP extended is actually limited in functionality:

- No streaming
- No possibility to add a stream in a 3GPP Extended File which progressive download has already started

SAF can be combined with the 3GPP extended file format in order to provide additional functionality while keeping backward compatibility.

An extension of the 3GPP file format will be required to:

– to make it efficient in streaming mode when a new stream is added ***in band***

– to limit ‘moov’/’moof’ parsing when it is not required, to reduce memory consumption

7.1.9.6 Integration in 3GPP 2 environment.

CMF components can be embedded in SAF streams to achieve compliance with 3GPP2.

7.2 The Mobile Open Rich-media Environment (MORE)

7.2.1 Overview

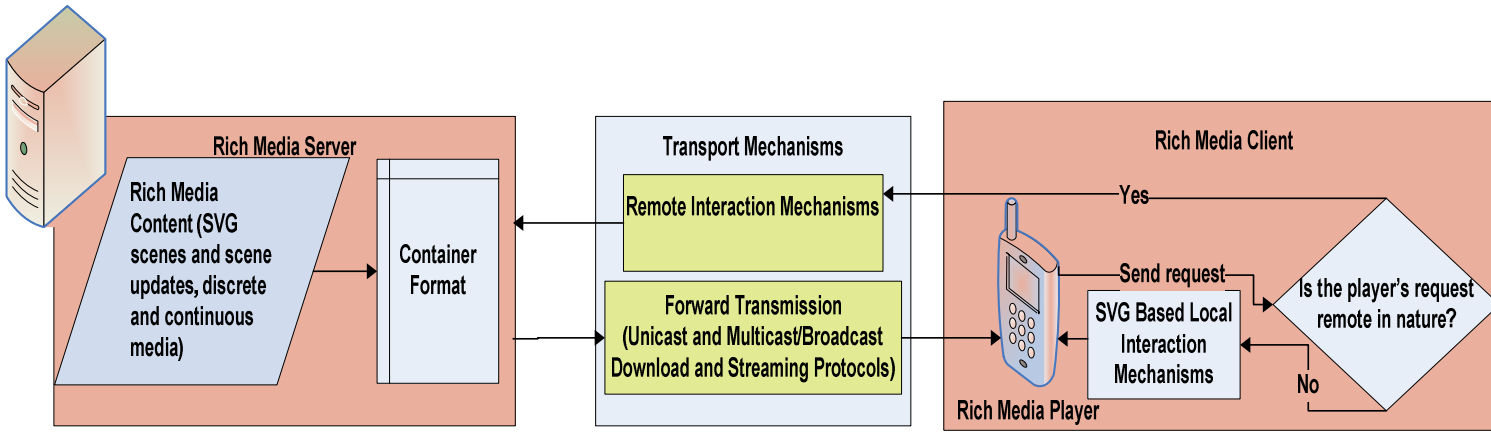


Figure 10: GENERAL ARCHITECTURE OF THE RICH MEDIA SYSTEM

The rich media system can be perceived as client-server architecture, comprising of 3 main components: The rich media server, transport mechanisms and the rich media client. Figure 1 illustrates the general architecture. The server takes as input, rich media content comprised of SVG, discrete (e.g. images) and continuous (e.g. audio, video) media. SVG content is represented as scenes and can be dynamically updated through scene updates. The rich media content can be encapsulated into a container format, containing additional information such as media synchronization, metadata, and hint tracks for packetization. The system then utilizes various transport mechanisms for 1-to-1 and 1-to-many protocols for download, progressive download and streaming scenarios as described in Section 9. The content is played on the client, allowing for local and remote interactivity of feedback and data requests. The MORE system is based on a non monolithic architecture emphasizing on a strong separation of interfaces and layers. This allows the flexibility of choosing the best of breed approach for a particular use case, and to change over time if necessary. It also minimizes the dependency on services in one layer to achieve performance in a higher layer.

7.2.2 Scene and Scene Updates

One of the motivations for rich media services is the ability to receive rich media content with minimal latency. In order to do so, the content or 'scene' on the client must be able to be dynamically updated with small changes rather than a completely new document being re-sent every time. Although SVG Mobile 1.2 supports prefetching for progressive downloading, during real-time streaming, a scene may change through animations and changes in scene states. This sequence of scene description and its spatial/temporal modifications needs to be streamed from the server to the players on the client device.

7.2.2.1 Scene

Scene describes the spatial organization of scene elements, the temporal organization of scene elements, synchronization information, and interaction among the elements. The scene presentation format and the rendering model are based on the Scalable Vector Graphics (SVG) format, a W3C Recommendation for representing two-dimensional graphics in XML language. Besides representation of graphics, SVG also supports a rich interaction based on DOM Level 3 Events and a complex animation model borrowed from SMIL specification.

A scene is typically first sent to the client to initialize the presentation layout. A scene can either be a complete SVG document or the content enclosed within <g></g> tags where the g element rendering will start when the g closing tag has been parsed and processed and when all internal and external resources required by the scene have been resolved. However, it is important to note that the initial scene delivered to the client to initialize the presentation and layout must be a complete and conforming SVG document. This helps the client to compute the initialization parameters such as 'viewport', 'viewbox',

and ‘aspectRatio’ of the rich media presentation. Further, a scene may use elements (<use>) previously defined in the <defs></defs> block or other objects within the scene that are not discarded by the client. This is similar to the prefetch functionality provided by SVG for progressive download.

7.2.2.2 Scene Updates

Scene updates refer to one or more incremental updates to the SVG Micro Document Object Model (uDOM) that get sent to the client device during streaming. These updates include element addition; element deletion; element replacement, element attribute updates and new scene operations. The new scene operation can be performed by replacing the entire SVG document or the logical scene that the user is visually engaged at that point in time i.e. typically the content enclosed with <g></g> tags as explained above.

Note that the updates can also be a combination of one or more of these operations depending on the desire of content provider. The client could potentially choose to update the SVG uDOM with this content update information without destroying and recreating the SVG uDOM for every streamed packet of information.

7.2.2.2.1 Scene Update syntax

The scene update syntax in MORE will follow the REX (Remote Events for XML) initiative in W3C that is spear-headed by SVG WG in an effort to meet the requirements of RME/DIMS specification. This is evident with the creation of a new Task Force (TF) in conjunction with Web Apps API WG to fast track this activity to meet the DIMS/RME requirements. The current draft specification is available at http://www.w3.org/TR/rex/ . The update syntax is compatible to the SVG and uDOM APIs as defined by SVG Mobile 1.2 specification.

The proposed XML update specification is be based on a set of requirements that are intended to maintain compatibility with DOM events, declarative in nature, and integrates well with the WWW architecture. The current charter of the Web Applications API will be responsible for maintaining this specification. Note that the syntax for update mechanism is not limited only to SVG but also extensible to other mark-ups, besides being very efficient and light weight for platforms that are already capable of supporting mobile SVG standard.

The following figure demonstrates the flow of delivering the scene and scene updates from the Rich Media server to the client, and the visual representation of the transmitted content at the client terminal.

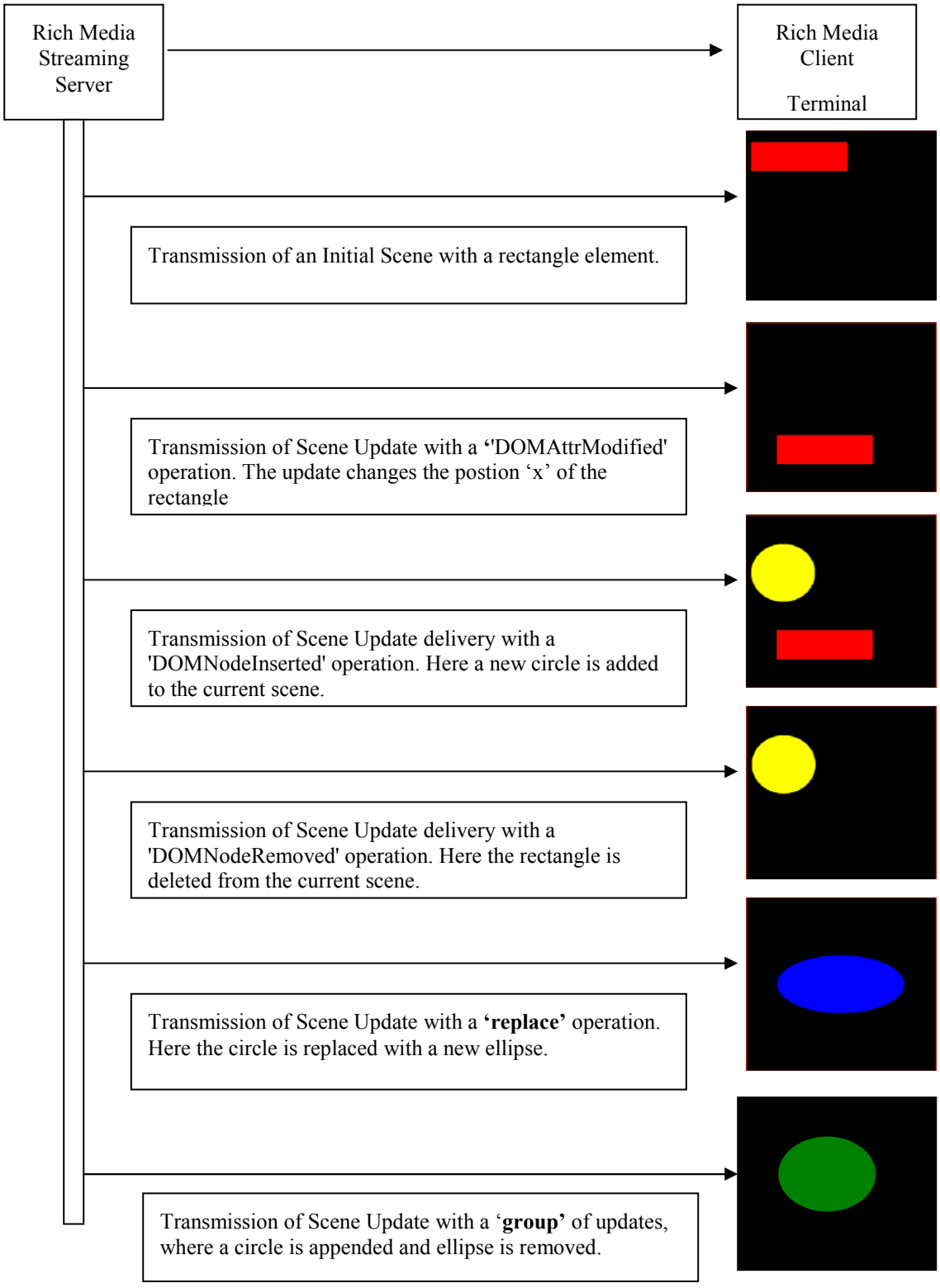


Figure 11: Illustration of Scene and Scene Updates delivery and realization.

7.2.2.3 Temporal Management of Scenes and Scene Updates

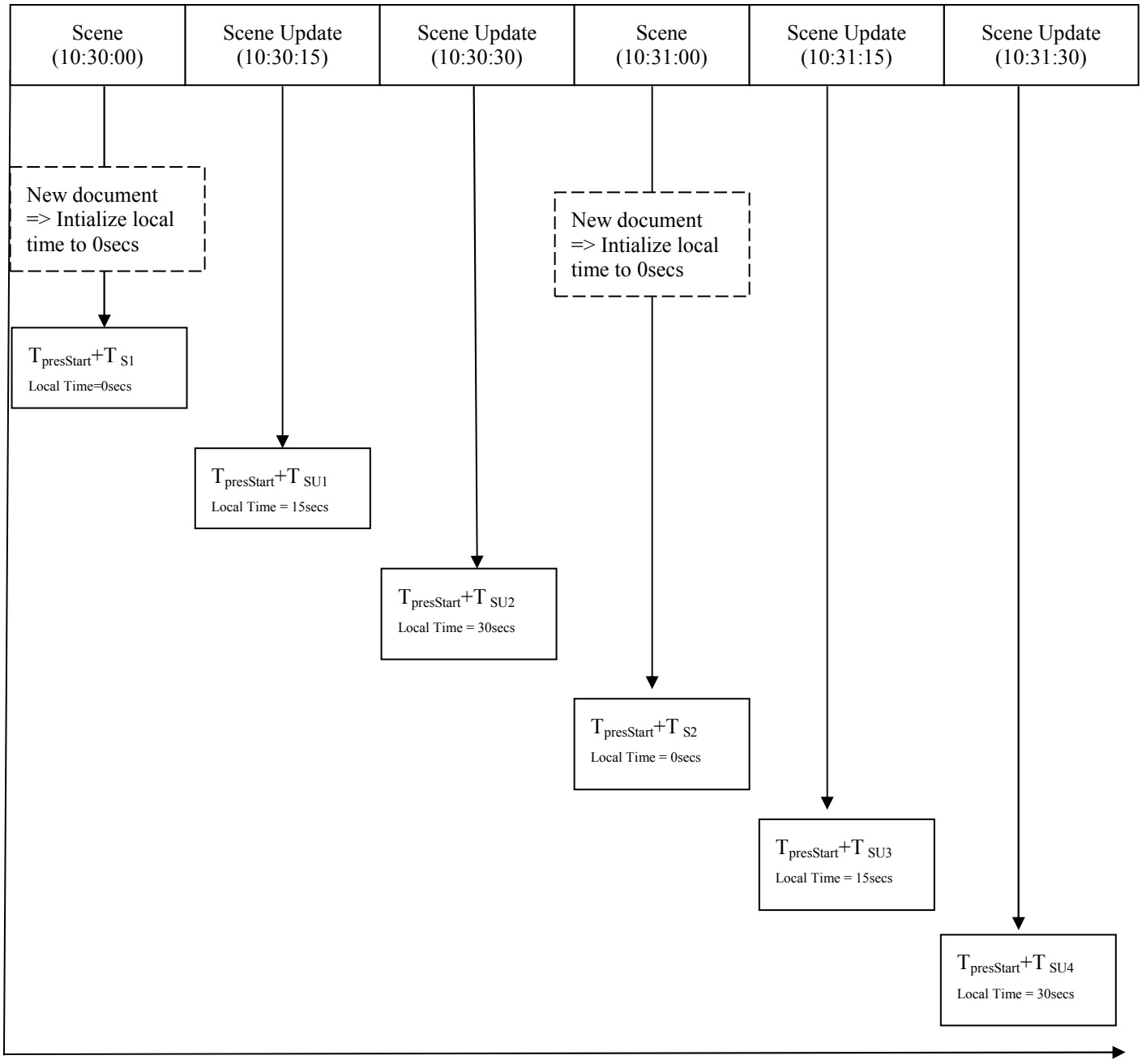
For temporal management, there is a need for an absolute rich media presentation start time denoted by $T_{presStart.}$ Every scene and scene update sample is associated with a timestamp relative to this overall rich media presentation start time ($T_{presSart}$). This relative scene/scene update timestamp refers to the actual rendering time or the time at which the scene/scene update is rendered on the client.

For example, if the corresponding timestamp of the first scene is T_{S1}, this is rendered at $T_{presStart} + T_{S1}$. Similarly if a succeeding scene update sample has a time stamp of T_{SU1}, it is rendered at $T_{presStart} + T_{SU1}$. However, if any scene or scene update sample arrives at the receiver at a time greater than it's rendering time, it may be ignored, simply rendered (in case of static objects) or retained for error concealment depending on the client's implementation capability. The retained sample can be utilized for the following scenarios:

1) Scene content playback. For e.g. the content at the client is required to playback based on user interaction from time zero of the presentation time.

2) Can be utilized for repairing the uDOM structure.

3) Error concealment based on client's needs.



Local time = local rendering time relative to current document timeline ----->

Figure 12: temporal management of scene and scene update

7.2.3 SVG and Associated Media

SVG supports media elements similar to Synchronized Multimedia Integration Language (SMIL) media elements. Continuous or real time media elements define their own timelines within their time container. All SVG media elements support the SVG timing attributes and run time synchronization.

7.2.3.1 Media Types

The media elements are audio, video and animation. However, particular platforms may have restrictions on the number of audio voices or channels that can be mixed, or the number of video streams that may be presented concurrently. Since these vary, the SVG language itself does not impose any such limits on audio or video.

7.2.3.2 Referencing Media

The real time media elements are audio and video, and are referenced as follows in SVG:

<xlink:href="example1.3gp" volume=".8" type="video/H264" x="10" y="170">

<xlink:href="example2.3gp"volume="0.7" type="audio/AMR-WB+" begin="mybutton.click" repeatCount="3">

Discrete media such as images are referenced in SVG using the 'image' element, such as:

<image x="200" y="200" width="100px" height="100px" xlink:href="myimage.png">

Furthermore, SVG can also reference other SVG documents, which in turn can reference yet more SVG documents through nesting. The referenced media elements can be linked through internal or external URLs in the SVG content. Here, internal URLs may refer to files internal to the SVG file or inline the host document, or within the container format. This applies to external URLs in a similar fashion.

The animation element specifies an external referenced SVG document or an SVG document fragment providing synchronized animated vector graphics. Like the video element, the animation element is a graphical object with size determined by its x, y, width and height attributes. For example:

<animation begin="1" dur="3" repeatCount="1.5" fill="freeze" x="100" y="100" xlink:href="myIcon.svg"/>

Also, SVG is capable of embedding media, like using Base64 encoding to embed images in the SVG file.

7.2.4 MORE Client Architecture

The MORE client is a lightweight entity present on the mobile terminal (Figure 4). This is substantiated due to the fact that it builds on top of existing application enablers such as SVG Mobile 1.2, ESMP, and XHTML-basic and thereby re-uses their associated underlying components such as the XML parser, rendering libraries, media decoders, and compression techniques. The client uses media packet depacketizers to obtain the different media that constitute the scene and scene updates in the case of real time streaming. In the case of download, the media is embedded media is internally (locally) or externally referenced. The synchronization module helps synchronize the frame rate and timing of continuous media with that of the non-frame based SVG content. The SVG engine in turn takes the different media and timing information as input to compose the dynamically rich multimedia presentation. The client is also responsible for transmitting any feedback occurring during interaction.

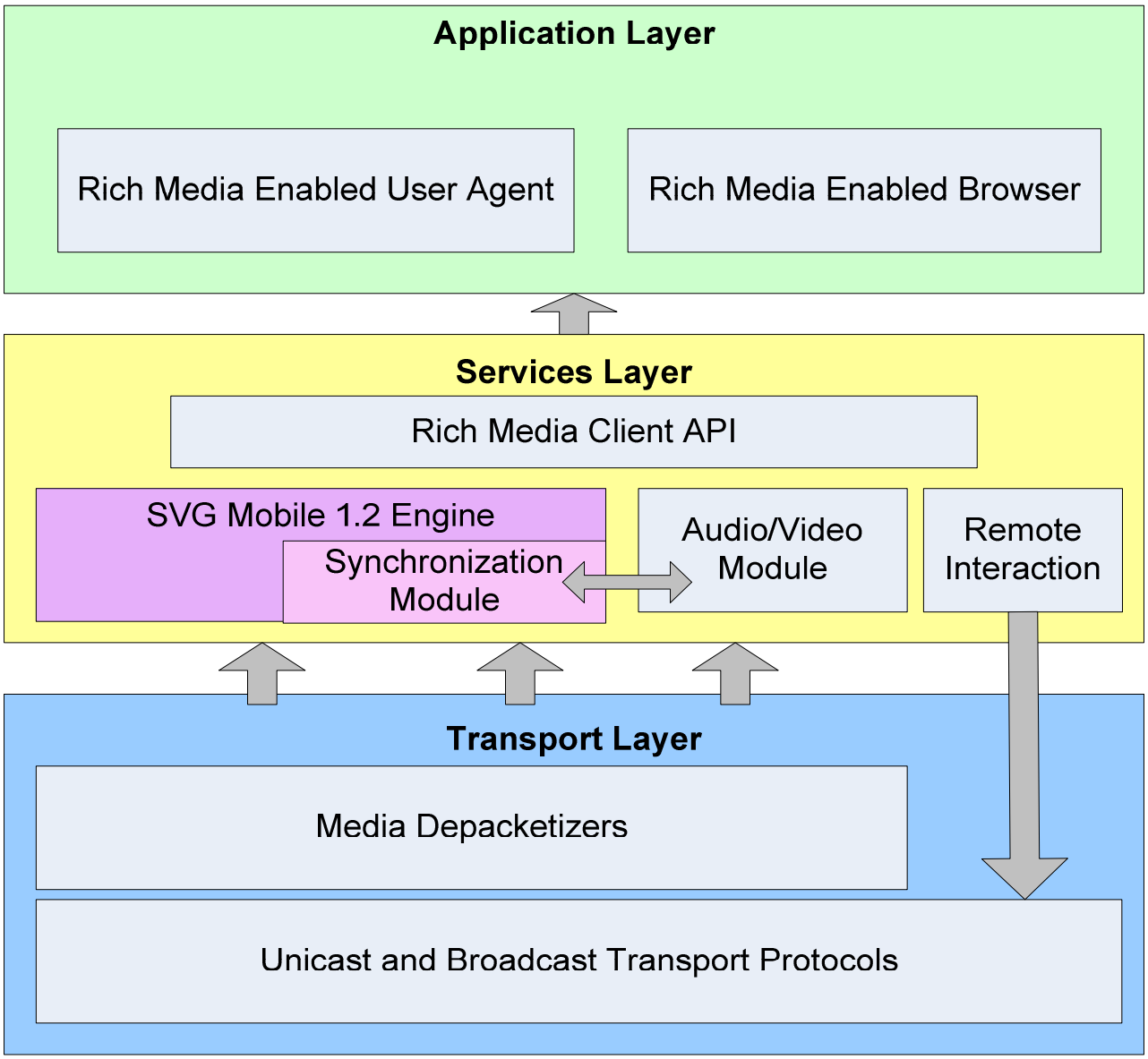


Figure 13: MORE Client Architecture

7.2.4.1 Synchronization Module

A rich media session comprises several media streams, and in the case of RTP, each is transported via a separate RTP session. Synchronization is first performed on the transport level (using RTP and NTP timestamps) and used as input to synchronization at the application level (SMIL based synchronization). At the application level, MORE utilizes the run-time synchronization functionality that SVG Mobile 1.2 inherits from SMIL 2.0 [13]. These attributes are syncBehavior, syncTolerance and syncMaster attributes, specified on the 'audio', 'video' and 'animation' elements, and syncBehaviorDefault and syncToleranceDefault attributes specified on the SVG element. For more information, please refer to Annex B.

Delivering media streams separately and resynchronizing them at the receiver, rather than being delivered bundled together has several inherent advantages: This model better reflects the preferences of the server or receiver. For example, in a video conference application, participants often prefer audio to video. In addition, different media can be assigned priority levels for

differing levels of error correction. In the case of bundled transport, all receivers would receive all media, which often is an issue for multiparty sessions using multicast distribution.

Annex C provides a detailed explanation of the synchronization process.

7.2.4.2 Resynchronization and Tune-In

During a rich media service, it is important for the clients to be able to connect and access the current streamed content, i.e. tune-in with minimal latency and data inaccuracy. MORE has several mechanisms to aid this purpose:

Random access points: An SVG SCENE or SCENE UPDATE for a new scene operation, can function as a random access point. Element add, delete, replace, and attribute update operations cannot function as random access points as they depend on previous content. A given SVG scene sample can be identified by the client to be a random access point by the random access bit (`A') set to 1 in the RTP payload header (section 7.2.6) or from the sync sample box (Section 7.2.5). Random access points are similar to INTRA frames in video. When tuning into a broadcast channel, the client can wait for the next random access point.

Time Synchronization: Packets of data received by the client are associated with timestamps relative to the overall presentation time container. Further, the sequence numbers associated with the packets determine the relative ordering. This information helps the client to decode and sample the data correctly and using discretion (e.g. ignoring late packets) regarding packets arriving later than the scheduled sample time.

7.2.4.3 Remote Interaction

During a rich media presentation, the client can request more information, update the content, or even send information back to the server. SVG provides local interaction through declarative animation and scripting. SVG Mobile 1.2 supports remote interaction via the Connection interface API [14] for socket-level communication. The API can be used for unicast based feedback over the HTTP/TCP protocol. Note that the Connection API in conjunction with uDOM support can also offer AJAX-like functionality.

Rather than advocating a particular transport mechanism for feedback, the MORE system focuses on a broader set of solutions particularly for remote interaction and mechanism for mapping local interactivity into remote commands for feedback and forward transmission. In the subsections below, details of the user events during interaction of rich media have been identified. These events are processed either locally or remotely, and be sent with either high delay or low delay, depending on the demand of application.

7.2.4.3.1 Locally and Remotely Processed Events

Application scripts used to process user interaction can be saved either on the client/UE side or on the server side, with the choice being application-specific.

Locally processed events are application scripts first processed on the client-side and if needed are transmitted to the server from the UE. For certain applications, scripts may be saved on the UE side. This can greatly reduce the burden of the server and facilitates the local interaction. For example, in interactive TV, manipulation of the user interface can be realized immediately at the UE side, and then some form of data can be sent to the server. In this case, the user may choose a channel; a script will process this event and send a PLAY request to the server. This request contains the information about the selected channel. Based on such information, the server may start a new broadcasting or downloading session to transmit the requested media data.

Remotely processed events are application scripts processed on the server side directly. In such a case, the user events are directly sent from the UE to the server without any initial processing. One possible reason for processing the data on the server side could be the security issue. The server in this case hides all details from the end user, so that the client only needs to report something like 'which button has been clicked by the user' or 'which text has been input by the user', and so on.

7.2.4.3.2 Generic Feedback Format

The SVG based feedback information is in the form of a text payload. The payload has two parts. The first part contains the MSG_ID, ELEMENT_ID and the EVENT, where the MSG_ID is a unique identifier to identify the feedback message from the client, ELEMENT_ID is the ID of the source element in the SVG DOM that triggers the event, and EVENT is an SVG event or a user defined event.

The actual feedback data is stored after the first part as a series of octets. This data may contain attributes of the SVG event itself [9].

For example the X and Y positions where the button was clicked may be directly transmitted to the server and the server can process the feedback remotely.

MSG_ID=1;ELEMENT_ID=”my-button1”;EVENT=”click”;[OCTET1OCTET2….OCTETN];

The above example consists of an SVG scene with a set of buttons to select a movie. On clicking one of the buttons, the client stores the X and Y positions where the button was clicked. This information is formulated into a remotely processed feedback message to the server. Octets store information such as clickX and clickY in this example. However, the actual feedback data can also contain the processed information like which movie the user selected. In this case the octets may contain information like “movieSelected= Lord of the Rings”. This is an example of a locally processed event. Therefore, on clicking one of the buttons in the scene, a script basically stores this value in a field called movieSelected. This information is formulated into a locally processed feedback message to the server. Note that the information or the stream of octets sent as part of the feedback payload is left to the discretion of the service or application.

Note that there is no particular restriction on the values of the octets in the feedback, but should follow a convention known to the service, i.e. the server and the clients.

7.2.4.4 Events and Event Management

The supported local events and their management in MORE are derived from SVG Mobile 1.2 and DOM Level 3 events model. They include DOM Events (focus, activate, mutation, etc), SVG Events (connection, load, etc.) and general XML events (user events, timing, key, and pointer events). For further information, please refer to Annex A.

7.2.4.5 Browser Interaction

Integration to the browser in MORE will follow and leverage the work of W3C Compound Documents Format (CDF) working group. The group is currently looking at various issues with combining multiple mark-up languages (XHTML+SVG, etc.) such as seamless event propagation, user interaction and rendering. MORE supports embedding SVG Mobile 1.2 content into XHTML document using the <object> tag through browser’s standard plug-in architecture. However, in the long term MORE shall be extensible to support the Compound document profiles, both CDR and CDI.

The Document Object Model (DOM) API supported in MORE is based on SVG DOM subset (uDOM) as defined in the SVG Mobile 1.2 specification. The motivation for uDOM support is to provide an API that allows dynamic manipulation of rich media content including modification of attribute and property values, creation of elements, event listener registration/removal and the ability to start or end media objects including animations, video, and audio. Scripting in MORE is handled via the ‘script’ element that contains executable content either through ESMP source code or compiled code such as Java (JAR archive that is compatible with JSR 226 API).

7.2.5 Container Format

SVG supports media elements similar to Synchronized Multimedia Integration Language (SMIL) media elements. The continuous media elements in particular, contain their own pre-defined frame based timing. The server is responsible for generating and transmitting packets containing rich media data to the clients in a temporally compliant manner with low delay request.

A container format would help in efficiently packaging the different media, providing timing synchronization, and enabling clients to realize, play, or render rich media content. The actual container used for rich media services, would however depend on the type of media (whether it is just SVG and XML based technologies, or contains other time based media such audio, video, etc.) and the nature of the application (download, progressive download, streaming for example).

Multipart MIME (MMIME) have recently taken on an important role in Web applications for HTTP based Unicast services. This MIME type defines how multiple data parts can be included within a single message. These parts can be regular text files, HTML documents, or binary data (such as images), where the multipart specification defines how these messages are combined together, as well as how binary data are encoded within the message. The different parts are placed in a single message, one after the other, separated by a special divider. This divider or boundary is a text string, defined in the MIME multipart content-type header field that precedes the entire message. This format is useful for download, and when time synchronization between media is not required.

ISO defines ISO Base Media File Format as a basis for developing a media container with various usages (download, progressive download and streaming). 3GPP and 3GPP2 derive file formats from the ISO Base File Format with differences being in the types of codecs supported in these formats. In MORE, we define some simple extensions the ISO Base Media File Format, conforming to the box semantics defined in it. This is only one of many choices provided for rich media services when a container format is needed. Provisions could then be explored on possible derivations to 3GPP and 3GPP2 file formats.

As of today, there are no solutions for embedding graphics media (SVG) into 3GPP ISO Base Media File Format, for progressive download or streaming of rich media content. Although previous work for transmitting a multimedia presentation comprising of several media objects within a container exist, the current solutions for vector graphics in 3GPP are only limited to download and play or otherwise known as HTTP streaming. MORE extends the file format's box hierarchy by adding relevant boxes to incorporate SVG as a new media. By adding an additional media track, leveraging the use of time synchronization along with existing audio and video track information, the solution is relatively simple and is extensible to other media formats if needed.

For more information, please refer to the MORE detailed proposal.

7.2.6 Transport Mechanisms

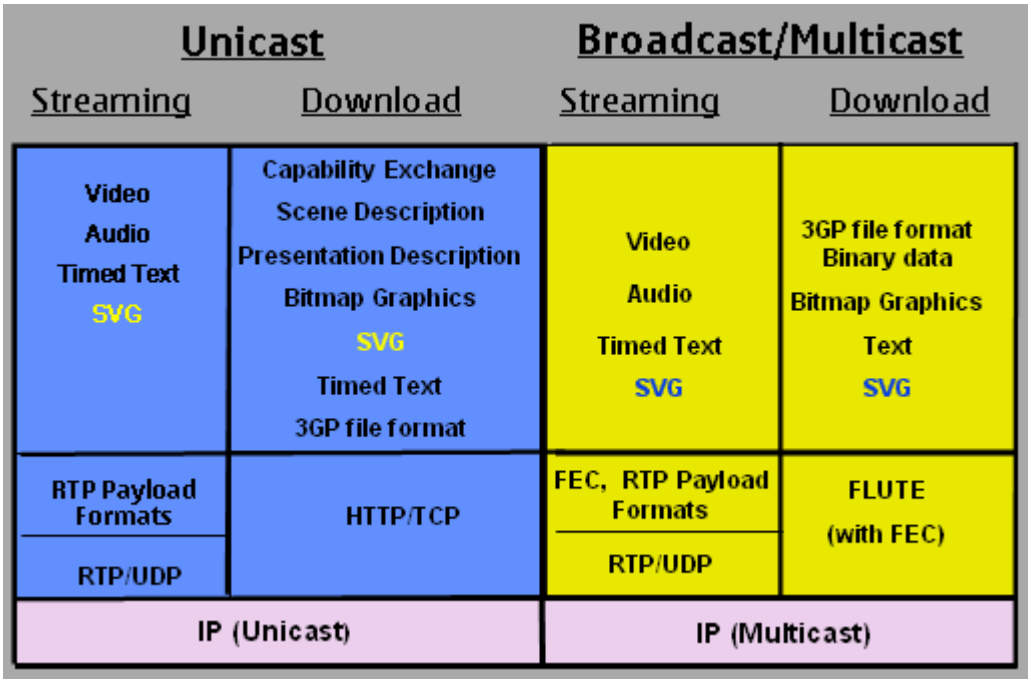


Figure 14: TRANSPORT SCENARIOS HANDLED BY MORE

7.2.6.1 Overview

The transport mechanisms support rich media delivery in the following modes: Unicast download (HTTP/TCP or MMS protocol), broadcast/multicast download (FLUTE/UDP), unicast streaming and broadcast/multicast streaming (RTP/UDP). For download mode, reliability is guaranteed by existing mechanisms in the transport and network layers, and no error resilience tools need to be designed at the application layer for rich media delivery. However, rich media transport in streaming mode is more challenging with UDP being unreliable. Therefore, the RTP design should provide some error resilience tools to help the media decoder cope up with unreliable transport.

SVG is traditionally considered to be a discrete media and hence no RTP payload format has been defined. It has been transported only in download and progressive download mode. With increasing richness and dynamism in the SVG presentations, it can now be considered as a continuous media. Consequently, we define an RTP payload format for SVG. Rich media is a combination of continuous media and discrete media, so rich media streaming should uses relevant transport mechanisms for these two media types. Rich media streaming is thus naturally realized by (a) streaming continuous media like SVG, video and audio (b) downloading the discrete media like images

The following sub-sections provide for a transport mechanism for supporting the download of SVG over FLUTE or the User Datagram Protocol (UDP). They also provide a specification of an RTP payload format that enables live streaming and the streaming of rich media content. Here, rich media content is encapsulated in RTP packets based upon the payload format at the sender.

For more information, please refer to the MORE detailed proposal.

7.2.7 Compression

The use of compression and content specific encoding techniques are economically driven decisions. Rich media content consists of SVG scenes and scene updates along with other referenced media. For streaming purposes, existing compression methods can be used for referenced media. However, compressing small sized SVG does not yield high benefits with the available bandwidth in today's networks. For large content, MORE recommends using Gzip as it results in high compression ratio. Hence, there is no specific need for introducing a new compression mechanism for rich media. Note however, that MORE does not preclude application of a specific encoding scheme that is widely adopted in the industry. This approach may be modified depending upon the outcome of the W3C work on XML compression as it tries to address compression for arbitrary XML data and not schema specific. In any case, it is important to view any encoding and compression decisions as orthogonal and separable from any base design decisions.

7.2.8 Conclusion

We present solutions that address various technology components needed for providing mobile real-time, interactive and streaming services. The solutions include dynamically delivering and updating scene content, a storage format for SVG content based on the ISO Base Media File Format including media synchronization, transport mechanisms and packetization for SVG and its discrete/continuous referenced media, and user interaction.

7.3 Summary

In the evaluation of the two main proposals received, namely those based on MPEG 4 part 20 and MORE, both can form the basis of meeting the RME requirements by meeting the vast majority of requirements directly allowing the residual requirements to be met through specificity in RME if the requirements remain necessary to be met.

Some common aspects have emerged from the proposals, for example:

- The scene description being based on SVG Tiny 1.2
- The need for any extensions beyond SVG Tiny 1.2 to be realized consistent with the rules of SVG.
- The need for a scene update mechanism
- The need to support streamed, e.g. broadcast, and non-streamed delivery.

However there are differences between the proposals, for example, the means to encode and convey the scenes and updates that need to be addressed.

Appendix A. Change History (Informative)

Document Identifier	Date	Sections	Description
OMA-WP-RME-20051004-d	02 Oct 2005	all	Initial draft providing the basic template
OMA-WP-Rich-Media-Environment-20051205-D	05 Dec 2005	1	Scope
OMA-WP-Rich-Media-Environment-20060127-D	27 Jan 2006	5.1	Presentation of MPEG4part20
OMA-WP-Rich-Media-Environment-20060310-D	10 Mar 2006	3.3	Abreviations and Annex1
OMA-WP-Rich-Media-Environment-20060315-D	15 Mar 2006	6	Inclusion of requirements and tables.
OMA-WP-Rich-Media-Environment-20060328-D	28 Mar 2006	7.1	Detailed description of MPEG4 part 20 proposal
OMA-WP-Rich-Media-Environment-20060406-D	06 Apr 2006	6 6 5.2 7.2 6	Inclusion of document 2006-153 Inclsion of document 362R04 Inclusion of document 2006-0103R03 Inclusion of document 2006-121R1 Inclusion of document 2006-0159R2
OMA-WP-Rich_Media_Environment-20060711-D	11 Jul 2006	4 7.3	Introduction and Summary sections completed Comments removed
OMA-WP-Rich_Media_Environment-20080408-D	09 Apr 2008	All	Editorial updates: - 2008 template and styles - History box fixed - Headers fixed
OMA-WP-Rich_Media_Environment-20081014-C	14 Oct 2008	All	Status changed to Candidate by TP: OMA-TP-2008-0376-INP_RME_V1_0_ERP_for_Candidate_Approval

Appendix B. example for the compliancy of LASeR with the XML processing model

Here is an example of a SAF + LASeR XML description of an application:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- authoring wrapper: does not exist in the binary -->
<saf:SAFSession xmlns:saf="urn:mpeg:mpeg4:SAF:2005"
                xmlns:xlink="http://www.w3.org/1999/xlink"
                xmlns="http://www.w3.org/2000/svg"
                xmlns:ev="http://www.w3.org/2001/xml-events"
                xmlns:lsr="urn:mpeg:mpeg4:LASeR:2005">

    <!-- first packet containing the encoding parameters -->
    <saf:sceneHeader>
        <lsr:LASeRHeader .../>
    </saf:sceneHeader>

    <!-- first scene packet containing a new scene
         the time information allows sending the packet in advance
         and presenting the updates at the right time -->
    <saf:sceneUnit time="0">
        <!-- the first packet of most scenes is a NewScene update
             containing the first "state" of the application -->
        <lsr:NewScene>
            <!-- simple SVG scene, complete, represents the initial state
                 of the application -->
            <svg id="root" width="180" height="177" viewBox="0 0 180 177">
                <rect id="rect" transform="translate(90 88)"
                      stroke="rgb(0,0,0)" fill="rgb(0,0,255)"
                      stroke-width="3" width="40" height="60"/>
            </svg>
        </lsr:NewScene>
    </saf:sceneUnit>

    <!-- list of updates to be executed at time 3s
         (3000 with default time resolution) -->
    <saf:sceneUnit time="3000">
        <lsr:Replace ref="#rect" attributeName="stroke"
                     value="rgb(255,0,0)"/>
        <!-- any number of updates can be placed here -->
    </saf:sceneUnit>

    <!-- update to be presented at 5s -->
    <saf:sceneUnit time="5000">
        <lsr:Replace ref="#rect" attributeName="fill"
                     value="rgb(127,51,204)"/>
    </saf:sceneUnit>

    <!-- any number of additional sceneUnit or other media headers
         or units can be placed here -->

    <!-- last packet of the application, signals that resources can be
         reclaimed -->
    <saf:endOfSAFSession/>

</saf:SAFSession>
```

Compared progressive rendering in SVG and LASeR

Example of a SVG file to be progressively rendered:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg width="176" height="144" viewBox="-4593 -100 9197 5749">
        <g fill="#B7DDC8" stroke="black" stroke-width="1">
                <rect id="reg00" x="-4593" y="-100" width="9197"
                        height="5749" fill="#EEEEEE"/>
                <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                <path id="reg10" fill="white" d="M2758 39691-12 0 0 -14 ..."/>
                <path id="reg02" fill="white" d="M1892 366112 -25 -13 ..."/>
                <path id="reg43" fill="white" d="M927 294510 -23 -14 ..."/>
                <path id="reg18" fill="white" d="M1767 21541-32 -39 0 ..."/>
                <path id="reg25" fill="white" d="M925 29451131 0 0 -9 ..."/>
                <path id="reg19" fill="white" d="M323 4070115 14 0 13 ..."/>
                <path id="reg04" fill="white" d="M123 27161762 -37 15 ..."/>
                <path id="reg26" fill="white" d="M523 1924173 35 14 7 ..."/>
                <path id="reg44" fill="white" d="M252 340910 216 31 36 ..."/>
                <path id="reg32" fill="white" d="M-1277 267912 74 0 885 ..."/>
                <path id="reg03" fill="white" d="M-3079 25091852 127 2 0 ..."/>
                <path id="reg05" fill="white" d="M-4404 1383127 -39 608 ..."/>
                <path id="reg29" fill="white" d="M-3360 1472150 9 404 ..."/>
                <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
                <path id="reg38" fill="white" d="M-3360 14721-409 -34 ..."/>
                <path id="reg13" fill="white" d="M-3310 1481127 -321 ..."/>
                <path id="reg48" fill="white" d="M-3838 11384 62 389 ..."/>
                <path id="reg27" fill="white" d="M-2327 11371-73 -76 ..."/>
                <path id="reg51" fill="white" d="M-1319 11001-1000 ..."/>
                <path id="reg35" fill="white" d="M-283 9651-59 -624 ..."/>
                <path id="reg42" fill="white" d="M-1319 92111036 44 ..."/>
                <path id="reg28" fill="white" d="M-1346 1881113 -372 ..."/>
                <path id="reg17" fill="white" d="M35 2094117 11 15 0 ..."/>
                <path id="reg06" fill="white" d="M-1346 18811283 6 3 ..."/>
                <path id="reg37" fill="white" d="M-1275 27531-2 -74 ..."/>
                <path id="reg24" fill="white" d="M-342 341159 624 0 ..."/>
                <path id="reg50" fill="white" d="M677 15641-110 -200 ..."/>
                <path id="reg16" fill="white" d="M567 13641110 200 ..."/>
                <path id="reg23" fill="white" d="M669 713127 25 29 ..."/>
                <path id="reg14" fill="white" d="M1244 1669173 560 ..."/>
                <path id="reg15" fill="white" d="M1290 16191375 -39 ..."/>
                <path id="reg36" fill="white" d="M2156 22571-14 0 0 ..."/>
                <path id="reg39" fill="white" d="M2744 18991-190 39 ..."/>
                <path id="reg33" fill="white" d="M3452 14511-75 -36 ..."/>
                <path id="reg20" fill="white" d="M3898 660171 344 29 ..."/>
                <path id="reg47" fill="white" d="M3350 2092171 -12 -19 ..."/>
                <path id="reg49" fill="white" d="M3021 197510 11 -17 ..."/>
                <path id="reg30" fill="white" d="M3840 671112 0 15 ..."/>
                <path id="reg22" fill="white" d="M3625 11441142 -23 ..."/>
                <path id="reg31" fill="white" d="M3452 14511-31 51 ..."/>
                <path id="reg07" fill="white" d="M3625 1451113 -13 ..."/>
                <path id="reg46" fill="white" d="M3552 7331271 -34 ..."/>
                <path id="reg40" fill="white" d="M3956 12501-14 -3 ..."/>
                <path id="reg21" fill="white" d="M3294 1787141 229 ..."/>
                <path id="reg01" fill="white" d="M-3017 49941-17 -13 ..."/>
                <path id="reg08" fill="white" d="M3350 17731-15 36 0 ..."/>
                <path id="reg12" fill="white" d="M-2152 4659116 -9 -..."/>
                <path id="reg09" fill="white" d="M3050 19781-32 62 ..."/>
        </g>
</svg>
```

Here is a proposed packetisation in SVG:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg width="176" height="144" viewBox="-4593 -100 9197 5749">
         <g fill="#B7DDC8" stroke="black" stroke-width="1">
                  <rect id="reg00" x="-4593" y="-100" width="9197"
                                 height="5749" fill="#EEEEEE"/>
                  <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                  <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                  <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                  <path id="reg10" fill="white" d="M2758 39691-12 0 0 -14 ..."/>
                  <path id="reg02" fill="white" d="M1892 366112 -25 -13 ..."/>
                  <path id="reg43" fill="white" d="M927 294510 -23 -14 ..."/>
                  <path id="reg18" fill="white" d="M1767 21541-32 -39 0 ..."/>
                  <path id="reg25" fill="white" d="M925 29451131 0 0 -9 ..."/>
                  <path id="reg19" fill="white" d="M323 4070115 14 0 13 ..."/>
                  <path id="reg04" fill="white" d="M123 27161762 -37 15 ..."/>
                  <path id="reg26" fill="white" d="M523 1924173 35 14 7 ..."/>
                  <path id="reg44" fill="white" d="M252 340910 216 31 36 ..."/>
                  <path id="reg32" fill="white" d="M-1277 267912 74 0 885 ..."/>
                  <path id="reg03" fill="white" d="M-3079 25091852 127 2 0 ..."/>
                  <path id="reg05" fill="white" d="M-4404 1383127 -39 608 ..."/>
                  <path id="reg29" fill="white" d="M-3360 1472150 9 404 ..."/>
                  <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
<!—SVG renderer will try to render here -->
                  <path id="reg38" fill="white" d="M-3360 14721-409 -34 ..."/>
                  <path id="reg13" fill="white" d="M-3310 1481127 -321 ..."/>
                  <path id="reg48" fill="white" d="M-3838 11384 62 389 ..."/>
                  <path id="reg27" fill="white" d="M-2327 11371-73 -76 ..."/>
                  <path id="reg51" fill="white" d="M-1319 11001-1000 ..."/>
                  <path id="reg35" fill="white" d="M-283 9651-59 -624 ..."/>
                  <path id="reg42" fill="white" d="M-1319 92111036 44 ..."/>
                  <path id="reg28" fill="white" d="M-1346 1881113 -372 ..."/>
                  <path id="reg17" fill="white" d="M35 2094117 11 15 0 ..."/>
                  <path id="reg06" fill="white" d="M-1346 18811283 6 3 ..."/>
                  <path id="reg37" fill="white" d="M-1275 27531-2 -74 ..."/>
                  <path id="reg24" fill="white" d="M-342 341159 624 0 ..."/>
                  <path id="reg50" fill="white" d="M677 15641-110 -200 ..."/>
                  <path id="reg16" fill="white" d="M567 13641110 200 ..."/>
                  <path id="reg23" fill="white" d="M669 713127 25 29 ..."/>
                  <path id="reg14" fill="white" d="M1244 1669173 560 ..."/>
                  <path id="reg15" fill="white" d="M1290 16191375 -39 ..."/>
                  <path id="reg36" fill="white" d="M2156 22571-14 0 0 ..."/>
<!—SVG renderer will try to render here -->
                  <path id="reg39" fill="white" d="M2744 18991-190 39 ..."/>
                  <path id="reg33" fill="white" d="M3452 14511-75 -36 ..."/>
                  <path id="reg20" fill="white" d="M3898 660171 344 29 ..."/>
                  <path id="reg47" fill="white" d="M3350 2092171 -12 -19 ..."/>
                  <path id="reg49" fill="white" d="M3021 197510 11 -17 ..."/>
                  <path id="reg30" fill="white" d="M3840 671112 0 15 ..."/>
                  <path id="reg22" fill="white" d="M3625 11441142 -23 ..."/>
                  <path id="reg31" fill="white" d="M3452 14511-31 51 ..."/>
                  <path id="reg07" fill="white" d="M3625 1451113 -13 ..."/>
                  <path id="reg46" fill="white" d="M3552 7331271 -34 ..."/>
                  <path id="reg40" fill="white" d="M3956 12501-14 -3 ..."/>
                  <path id="reg21" fill="white" d="M3294 1787141 229 ..."/>
                  <path id="reg01" fill="white" d="M-3017 49941-17 -13 ..."/>
                  <path id="reg08" fill="white" d="M3350 17731-15 36 0 ..."/>
                  <path id="reg12" fill="white" d="M-2152 4659116 -9 -..."/>
                  <path id="reg09" fill="white" d="M3050 19781-32 62 ..."/>
         </g>
</svg>
<!—SVG renderer will definitely render here -->
```

At each rendering points, the SVG UA renders a non-well formed XML tree. At the first rendering point, the current tree is:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg width="176" height="144" viewBox="-4593 -100 9197 5749">
         <g fill="#B7DDC8" stroke="black" stroke-width="1">
                  <rect id="reg00" x="-4593" y="-100" width="9197"
                                 height="5749" fill="#EEEEEE"/>
                  <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                  <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                  <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                  <path id="reg10" fill="white" d="M2758 39691-12 0 0 -14 ..."/>
                  <path id="reg02" fill="white" d="M1892 366112 -25 -13 ..."/>
                  <path id="reg43" fill="white" d="M927 294510 -23 -14 ..."/>
                  <path id="reg18" fill="white" d="M1767 21541-32 -39 0 ..."/>
                  <path id="reg25" fill="white" d="M925 29451131 0 0 -9 ..."/>
                  <path id="reg19" fill="white" d="M323 4070115 14 0 13 ..."/>
                  <path id="reg04" fill="white" d="M123 27161762 -37 15 ..."/>
                  <path id="reg26" fill="white" d="M523 1924173 35 14 7 ..."/>
                  <path id="reg44" fill="white" d="M252 340910 216 31 36 ..."/>
                  <path id="reg32" fill="white" d="M-1277 267912 74 0 885 ..."/>
                  <path id="reg03" fill="white" d="M-3079 25091852 127 2 0 ..."/>
                  <path id="reg05" fill="white" d="M-4404 1383127 -39 608 ..."/>
                  <path id="reg29" fill="white" d="M-3360 1472150 9 404 ..."/>
                  <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
```

Which violates the XML model.

LASeR would do this:

```
<?xml version="1.0" encoding="UTF-8"?>
<saf:SAFSession xmlns:saf="urn:mpeg:mpeg4:SAF:2005"
    xmlns:lsr="urn:mpeg:mpeg4:LASeR:2005" xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <saf:sceneHeader>
        <lsr:LASeRHeader .../>
    </saf:sceneHeader>
    <saf:sceneUnit>
        <lsr:NewScene>
          <svg width="176" height="144"
               viewBox="-4593 -100 9197 5749">
               <g id="root" fill="#B7DDC8" stroke="black" stroke-width="1">
                   <rect id="reg00" x="-4593" y="-100" width="9197"
                                 height="5749" fill="#EEEEEE"/>
                   <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                   <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                   <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                   <path id="reg10" fill="white" d="M2758 39691-12 0 0 -14 ..."/>
                   <path id="reg02" fill="white" d="M1892 366112 -25 -13 ..."/>
                   <path id="reg43" fill="white" d="M927 294510 -23 -14 ..."/>
                   <path id="reg18" fill="white" d="M1767 21541-32 -39 0 ..."/>
                   <path id="reg25" fill="white" d="M925 29451131 0 0 -9 ..."/>
                   <path id="reg19" fill="white" d="M323 4070115 14 0 13 ..."/>
                   <path id="reg04" fill="white" d="M123 27161762 -37 15 ..."/>
                   <path id="reg26" fill="white" d="M523 1924173 35 14 7 ..."/>
                   <path id="reg44" fill="white" d="M252 340910 216 31 36 ..."/>
                   <path id="reg32" fill="white" d="M-1277 267912 74 0 885 ..."/>
                   <path id="reg03" fill="white" d="M-3079 25091852 127 2 0 ..."/>
                   <path id="reg05" fill="white" d="M-4404 1383127 -39 608 ..."/>
                   <path id="reg29" fill="white" d="M-3360 1472150 9 404 ..."/>
                   <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
               </g>
           </svg>
        </lsr:NewScene>
    </saf:sceneUnit>

<!-- other packets here -->

    <saf:endOfSAFSession/>

</saf:SAFSession>
```

The SVG content in the LASeR first packet is highlighted in blue. It is a complete and well-formed subtree. It obeys the XML model.

The complete content in LASeR would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<saf:SAFSession xmlns:saf="urn:mpeg:mpeg4:SAF:2005"
    xmlns:lsr="urn:mpeg:mpeg4:LASeR:2005" xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink">
    <saf:sceneHeader>
        <lsr:LASeRHeader .../>
    </saf:sceneHeader>
    <saf:sceneUnit>
        <lsr:NewScene>
          <svg width="176" height="144"
               viewBox="-4593 -100 9197 5749">
              <g id="root" fill="#B7DDC8" stroke="black" stroke-width="1">
                  <rect id="reg00" x="-4593" y="-100" width="9197"
                                height="5749" fill="#EEEEEE"/>
                  <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                  <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                  <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                  <path id="reg10" fill="white" d="M2758 39691-12 0 0 -14 ..."/>
                  <path id="reg02" fill="white" d="M1892 366112 -25 -13 ..."/>
                  <path id="reg43" fill="white" d="M927 294510 -23 -14 ..."/>
                  <path id="reg18" fill="white" d="M1767 21541-32 -39 0 ..."/>
                  <path id="reg25" fill="white" d="M925 29451131 0 0 -9 ..."/>
                  <path id="reg19" fill="white" d="M323 4070115 14 0 13 ..."/>
                  <path id="reg04" fill="white" d="M123 27161762 -37 15 ..."/>
                  <path id="reg26" fill="white" d="M523 1924173 35 14 7 ..."/>
                  <path id="reg44" fill="white" d="M252 340910 216 31 36 ..."/>
                  <path id="reg32" fill="white" d="M-1277 267912 74 0 885 ..."/>
                  <path id="reg03" fill="white" d="M-3079 25091852 127 2 0 ..."/>
                  <path id="reg05" fill="white" d="M-4404 1383127 -39 608 ..."/>
                  <path id="reg29" fill="white" d="M-3360 1472150 9 404 ..."/>
                  <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
              </g>
            </svg>
        </lsr:NewScene>
    </saf:sceneUnit>

    <saf:sceneUnit time="1000">
        <lsr:Insert id=”root”>
                  <path id="reg38" fill="white" d="M-3360 14721-409 -34 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg13" fill="white" d="M-3310 1481127 -321 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg48" fill="white" d="M-3838 11384 62 389 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg27" fill="white" d="M-2327 11371-73 -76 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg51" fill="white" d="M-1319 11001-1000 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg35" fill="white" d="M-283 9651-59 -624 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg42" fill="white" d="M-1319 92111036 44 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                  <path id="reg28" fill="white" d="M-1346 1881113 -372 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
```

```
                <path id="reg17" fill="white" d="M35 2094117 11 15 0 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg06" fill="white" d="M-1346 18811283 6 3 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg37" fill="white" d="M-1275 27531-2 -74 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg24" fill="white" d="M-342 341159 624 0 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg50" fill="white" d="M677 15641-110 -200 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg16" fill="white" d="M567 13641110 200 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg23" fill="white" d="M669 713127 25 29 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg14" fill="white" d="M1244 1669173 560 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg15" fill="white" d="M1290 16191375 -39 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg36" fill="white" d="M2156 22571-14 0 0 ..."/>
        </lsr:Insert>
    </saf:sceneUnit>

    <saf:sceneUnit time="2000">
        <lsr:Insert id=”root”>
                <path id="reg39" fill="white" d="M2744 18991-190 39 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg33" fill="white" d="M3452 14511-75 -36 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg20" fill="white" d="M3898 660171 344 29 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg47" fill="white" d="M3350 2092171 -12 -19 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg49" fill="white" d="M3021 197510 11 -17 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg30" fill="white" d="M3840 671112 0 15 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg22" fill="white" d="M3625 11441142 -23 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg31" fill="white" d="M3452 14511-31 51 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg07" fill="white" d="M3625 1451113 -13 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg46" fill="white" d="M3552 7331271 -34 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg40" fill="white" d="M3956 12501-14 -3 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg21" fill="white" d="M3294 1787141 229 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                <path id="reg01" fill="white" d="M-3017 49941-17 -13 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
```

```
                    <path id="reg08" fill="white" d="M3350 17731-15 36 0 ..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                    <path id="reg12" fill="white" d="M-2152 4659116 -9 -..."/>
        </lsr:Insert>
        <lsr:Insert id=”root”>
                    <path id="reg09" fill="white" d="M3050 19781-32 62 ..."/>
        </lsr:Insert>
    </saf:sceneUnit>

    <saf:endOfSAFSession/>

</saf:SAFSession>
```

The scene tree in the SVG UA after the first LASeR packet has been received and decoded, is:

```
          <svg id="root" width="176" height="144"
               viewBox="-4593 -100 9197 5749">
              <g fill="#B7DDC8" stroke="black" stroke-width="1">
                    <rect id="reg00" x="-4593" y="-100" width="9197"
                                height="5749" fill="#EEEEEE"/>
                    <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                    <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                    <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                    ...
                    <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
            </g>
          </svg>
```

The scene tree in the SVG UA after the second LASeR packet has been received and decoded, is:

```
      <svg id="root" width="176" height="144"
               viewBox="-4593 -100 9197 5749">
              <g fill="#B7DDC8" stroke="black" stroke-width="1">
                    <rect id="reg00" x="-4593" y="-100" width="9197"
                                height="5749" fill="#EEEEEE"/>
                    <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
                    <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
                    <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
                    ...
                    <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
                    <path id="reg38" fill="white" d="M-3360 14721-409 -34 ..."/>
                    <path id="reg13" fill="white" d="M-3310 1481127 -321 ..."/>
                    <path id="reg48" fill="white" d="M-3838 11384 62 389 ..."/>
                    ...
                    <path id="reg36" fill="white" d="M2156 22571-14 0 0 ..."/>
            </g>
      </svg>
```

The scene tree in the SVG UA after the third LASeR packet has been received and decoded, is:

```
<svg id="root" width="176" height="144"
          viewBox="-4593 -100 9197 5749">
          <g fill="#B7DDC8" stroke="black" stroke-width="1">
              <rect id="reg00" x="-4593" y="-100" width="9197"
                          height="5749" fill="#EEEEEE"/>
              <path id="reg34" fill="white" d="M2283 252511038 -114 31..."/>
              <path id="reg41" fill="white" d="M2542 33421-13 0 0 12 ..."/>
              <path id="reg11" fill="white" d="M1894 36361-2 25 16 14 ..."/>
              ...
              <path id="reg45" fill="white" d="M-2369 15571-537 -48 ..."/>
              <path id="reg38" fill="white" d="M-3360 14721-409 -34 ..."/>
              <path id="reg13" fill="white" d="M-3310 1481127 -321 ..."/>
              <path id="reg48" fill="white" d="M-3838 11384 62 389 ..."/>
              ...
              <path id="reg36" fill="white" d="M2156 22571-14 0 0 ..."/>
              <path id="reg39" fill="white" d="M2744 18991-190 39 ..."/>
              <path id="reg33" fill="white" d="M3452 14511-75 -36 ..."/>
              <path id="reg20" fill="white" d="M3898 660171 344 29 ..."/>
              ...
              <path id="reg09" fill="white" d="M3050 19781-32 62 ..."/>
          </g>
</svg>
```

Appendix C. Annex A: SVG EVENTS

This section contains the list the supported events as specified in the SVG Mobile 1.2 draft specification: http://www.w3.org/TR/SVGMobile12/interact.html#SVGEvents.

Event Identifier {event-namespace, event-localname}	Description	DOM3 event category	Animation event name	uDOM interface
{"http://www.w3.org/2001/xml-events", **"DOMFocusIn"**} SVG 1.2 alias: {"http://www.w3.org/2001/xml-events", "focusin"} (see Notes below).	Occurs when an element receives focus.	UIEvent	focusin	`UIEvent`
{"http://www.w3.org/2001/xml-events", **"DOMFocusOut"**} SVG 1.2 alias: {"http://www.w3.org/2001/xml-events", "focusout"} (see Notes below).	Occurs when an element loses focus.	UIEvent	focusout	`UIEvent`
{"http://www.w3.org/2001/xml-events", **"DOMActivate"**} SVG 1.2 alias: {"http://www.w3.org/2001/xml-events", "activate"} (see Notes below).	Occurs when an element is activated, for instance, thru a mouse click or a keypress	UIEvent	activate	`UIEvent`
{"http://www.w3.org/2001/xml-events", **"click"**}	Occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is: `mousedown`, `mouseup`, `click`.	MouseEvent	click	`MouseEvent`
{"http://www.w3.org/2001/xml-events", **"mousedown"**}	Occurs when the pointing device button is pressed over an element.	MouseEvent	mousedown	`MouseEvent`
{"http://www.w3.org/2001/xml-events", **"mouseup"**}	Occurs when the pointing device button is released over an element.	MouseEvent	mouseup	`MouseEvent`

{"http://www.w3.org/2001/xml-events", **"mouseover"**}	Occurs when the pointing device is moved onto an element.	MouseEvent	mouseover	MouseEvent
{"http://www.w3.org/2001/xml-events", **"mousemove"**}	Occurs when the pointing device is moved while it is over an element.	MouseEvent	mousemove	MouseEvent
{"http://www.w3.org/2001/xml-events", **"mouseout"**}	Occurs when the pointing device is moved away from an element.	MouseEvent	mouseout	MouseEvent
{"http://www.w3.org/2001/xml-events", **"textInput"**}	One or more characters have been entered.	TextEvent	none	TextEvent
{"http://www.w3.org/2001/xml-events", **"keydown"**}	A key is pressed down. (The normative definition of this event is the description in the DOM3 Events specification.)	KeyboardEvent	none	KeyboardEvent
{"http://www.w3.org/2001/xml-events", **"keyup"**}	A key is released. (The normative definition of this event is the description in the DOM3 Events specification.)	KeyboardEvent	none	KeyboardEvent
{"http://www.w3.org/2001/xml-events", **"load"**} Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGLoad"} (see Notes below).	The event is triggered at the point at which the user agent has fully parsed the element and its descendants and is ready to act appropriately upon that element, such as being ready to render the element to the target device. Referenced external resources that are required must be loaded, parsed and ready to render before the event is triggered. Optional external resources are not required to be ready for the event to be triggered.	HTMLEvent	load	Event

{"http://www.w3.org/2001/xml-events", **"resize"**} Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGResize"} (see Notes below).	Occurs when a document view is being resized. This event is only applicable to **'svg'** elements and is dispatched after the resize operation has taken place. The target of the event is the **'svg'** element.	HTMLEvent	resize	Event
{"http://www.w3.org/2001/xml-events", **"scroll"**} Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGScroll"} (see Notes below).	Occurs when a document view is being shifted along the X or Y or both axis, either through a direct user interaction or any change on the **'currentTranslate'** property available on SVGSVGElement interface. This event is only applicable to **'svg'** elements and is dispatched after the shift modification has taken place. The target of the event is the **'svg'** element.	HTMLEvent	scroll	Event
{"http://www.w3.org/2001/xml-events", **"zoom"**} Deprecated backwards-compatibility alias: {"http://www.w3.org/2001/xml-events", "SVGZoom"} (see Notes below).	Occurs when the zoom level of a document view is being changed, either through a direct user interaction or any change to the **'currentScale'** property available on SVGSVGElement interface. This event is only applicable to **'svg'** elements and is dispatched after the zoom level modification has taken place. The target of the event is the **'svg'** element.	DOM3's SVG Events	zoom	Event
{"http://www.w3.org/2001/xml-events", **"beginEvent"**}	Occurs when an animation element begins. For details, see the description of the Events and event model in SMIL 2.0.	DOM3's Timing Events	beginEvent	TimeEvent

{"http://www.w3.org/2001/xml-events", **"endEvent"**}.	Occurs when an animation element ends. For details, see the description of the Events and event model in SMIL 2.0.	DOM3's Timing Events	endEvent	TimeEvent
{"http://www.w3.org/2001/xml-events", **"repeatEvent"**}	Occurs when an animation element repeats. It is raised each time the element repeats, after the first iteration. For details, see the description of the Events and event model in SMIL 2.0.	DOM3's Timing Events	repeat	TimeEvent
{"http://www.w3.org/2001/xml-events", **"wheel"**}	Occurs when a rotational input device has been activated.	UIEvent	none	WheelEvent
{"http://www.w3.org/2000/svg", **"preload"**}	A load operation has begun.	none	none	ProgressEvent
{"http://www.w3.org/2000/svg", **"loadProgress"**}	Progress has occurred in loading a given resource.	none	none	ProgressEvent
{"http://www.w3.org/2000/svg", **"postload"**}	A load operation has completed.	none	none	ProgressEvent
{"http://www.w3.org/2001/xml-events", **"timer"**}	Occurs when the specified timer interval has elapsed for a timer. This event is triggered only by 'enabled' timers in the current global execution context of the SVG document (i.e. for timers which have been instantiated via the SVGGlobal interface and started via the start() method of the SVGTimer interface).	none	none	Event
{"http://www.w3.org/2000/svg", **"connectionConnected"**}	Occurs when a connection has been established. No context information is available.	none	none	ConnectionEvent
{"http://www.w3.org/2000/svg", **"connectionClosed"**}	Occurs when a connection has been closed. No context information is available	none	none	ConnectionEvent

{"http://www.w3.org/2000/svg", **"connectionError"**}	Occurs when an error happens during the lifetime of a connection. Additional context information is available in the errorCode field.	none	none	ConnectionEvent
{"http://www.w3.org/2000/svg", **"connectionDataSent"**}	Occurs when data has been successfully transmitted. No context information is available.	none	none	ConnectionEvent
{"http://www.w3.org/2000/svg", **"connectionDataReceived"**}	Occurs when data has been received on the connection. Additional context information is available on the receivedData field.	none	none	ConnectionEvent

Appendix D. Annex B: APPLICATION LEVEL SYNCHRONIZATION

The follow section describes the application level synchronization as specified by the run-time synchronization. http://www.w3.org/TR/SVGMobile12/multimedia.html#Smil2Sync .

(a) syncBehavior = (canSlip | locked | independent | default)

Defines the runtime synchronization behavior for an element.

Legal values are:

canSlip: Allows the associated element to slip with respect to the parent time container. When this value is used, any syncTolerance attribute is ignored.

Locked: Forces the associated element to maintain sync with respect to the parent time container. This can be eased with the use of the syncTolerance attribute.

Independent: Declares an independent timeline that is scheduled with the timegraph, but will ignore any seek operations on the parent.

Default: The runtime synchronization behavior for the element is determined by the value of the syncBehaviorDefault attribute. This is the default value.

The argument value independent is equivalent to setting syncBehavior="canSlip" and syncMaster="true" so that the element is scheduled within the timegraph, but is unaffected by any other runtime synchronization issues. Setting syncBehavior="canSlip" and syncMaster="true" declares the element as being the synchronization master clock and that the element may slip against its parent time line

(b) syncTolerance = (Clock-value | default)

This attribute on timed elements and time containers defines the synchronization tolerance for the associated element. The attribute has an effect only if the element's runtime synchronization behavior is "locked". This allows a locked sync relationship to ignore a given amount of slew without forcing resynchronization.

Clock-value: Specifies the synchronization tolerance as a value. Clock values are measured in element simple time.

Default: The synchronization tolerance for the element is determined by the value of the syncToleranceDefault attribute. This is the default value.

(c) syncMaster

Boolean attribute on media elements and time containers that forces other elements in the time container to synchronize their playback to this element. The default value is false. The associated property is read-only, and cannot be set by script.

Controlling the default behavior

Two attributes are defined to specify the default behavior for runtime synchronization:

(d) syncBehaviorDefault = (canSlip | locked | independent | inherit)

Defines the default value for the runtime synchronization behavior for an element. The values "canSlip", "locked" and "independent" specify that the element's runtime synchronization behavior is the respective value.

Inherit: Specifies that the value of this attribute (and the value of the element's runtime synchronization behavior) are inherited from the syncBehaviorDefault value of the parent element. If there is no parent element, the value is implementation dependent. This is the default value.

(e) syncToleranceDefault = (Clock-value | inherit)

Defines the default value for the runtime synchronization tolerance value for an element. Clock values specify that the element's runtime synchronization tolerance value is the respective value.

Inherit: Specifies that the value of this attribute (and the value of the element's runtime synchronization tolerance value) are inherited from the syncToleranceDefault value of the parent element. If there is no parent element, the value is implementation dependent but should be no greater than two seconds. This is the default value.

The accumulated synchronization offset

If an element slips synchronization relative to its parent, the amount of this slip at any point is described as the *accumulated synchronization offset*. This offset is used to account for pause semantics as well as performance or delivery related slip. This value is used to adjust the conversion between element and parent times, as described in Converting between local and global times. The offset is computed as follows:

Let $\mathbf{t_c(t_{ps})}$ be the computed element active time for an element at the parent simple time $\mathbf{t_{ps}}$, according to the defined synchronization relationship for the element.

Let $\mathbf{t_o(t_{ps})}$ be the observed element active time for an element at the parent simple time $\mathbf{t_{ps}}$.

The accumulated synchronization offset **O** is:

$$\mathbf{O = t_o(t_{ps}) - t_c(t_{ps})}$$

This offset is measured in parent simple time.

Thus an accumulated synchronization offset of 1 second corresponds to the element playing 1 second "later" than it was scheduled. An offset of -0.5 seconds corresponds to the element playing a half second "ahead" of where it should be.

Appendix E. Annex C: TRANSPORT LEVEL SYNCHRONIZATION

As explained in the transport section, rich media streaming is realized by (a) streaming continuous media like SVG, video and audio using RTP (b) downloading the discrete media like images over FLUTE.

In both RTP-RTP and RTP-FLUTE synchronization scenarios, one media is played in-sync with the other media. In the case of an RTP-FLUTE combination, the discrete media is generally played in-sync with one of the continuous media, whereas in the RTP-RTP case, both are continuous media and the reference media depends largely on the application.

The RTP-FLUTE case is more straightforward as the discrete media chunks tend to have longer playout times than the continuous media chunks (e.g. 20 ms audio frames or 100 ms video frames). The FLUTE packets are transmitted in advance so that the receiver can make sure it has the entire discrete media chunk reconstructed from the FLUTE packets (some of which might be lost or arrive out of order). Also information is needed to inform the receiver which discrete media chunk has to be rendered in sync with a particular segment of the continuous media. In SVG data, the lifecycle of the discrete media can be defined to aid in this synchronization and also, scripts can be sent in advance to the receiver to provide more synchronization information if needed.

In the RTP-RTP case, the server enables synchronization of media streams at the receiver by running a common presentation based reference clock and periodically announcing through RTCP, the relationship between the reference clock time and the media stream time. As the reference clock runs at a constant rate, correspondence points between the reference clock and the media stream allow the receiver to calculate the relative timing relationship between the media streams.

The correspondence between the reference clock and the media clock is noted when each RTCP packet is generated. An offset is then calculated between the individual media times and reference clock. The common reference clock is the “wall clock” time used by RTCP. It takes the form of an NTP-format timestamp, counting seconds and fractions of a second since midnight UTC (coordinated Universal Time) on January 1, 1900. The server periodically establishes a correspondence between the media clock for each stream and the common reference clock; communicated to receivers via RTCP sender report packets.

Synchronized clocks are required only when media streams generated by different hosts are being synchronized. Also, it is necessary to identify specific media streams/sources that need to be synchronized. RTP provides this information through synchronization source identifiers (SSRC), giving the related sources a shared name to distinguish streams to be synchronized from the independent ones. A mapping from SSRC identifiers to a persistent canonical name (CNAME) is provided by the RTCP source description (SDES) packets. A sender should ensure that RTP sessions to be synchronized on playout have a common CNAME so that receivers know how to align media. The CNAME is determined algorithmically according to the user name and network address of the source host. In the case of multiple hosts, the RTP standard requires each host to use its own IP address as part of the CNAME.

The receiver (rich media client) synchronizes those streams that the sender has given the same CNAME in their RTCP source description packets. The actual synchronization process is triggered by the reception of RTCP sender report packets containing the mapping between the media clock and a reference clock common to all the media. Once this mapping has been determined for the media streams, the receiver has the information needed to synchronize playout.

The first step of synchronization is to determine, for each stream to be synchronized, when the media data corresponding to a particular reference time is to be presented to the user. Due to several reasons including network latency, two streams transmitted at the same time may not be scheduled for presentation at the same time if the playout times are determined independently. Thus, the playout time for one stream has to be adjusted to match the other. This adjustment translates into an offset to be added to the playout buffering delay for one stream, such that the media are played out in time alignment.

The receiver observes the mapping between the media clock and the reference clock as assigned by the sender, for each media stream it is to synchronize. This mapping is conveyed to the receiver in periodic RTCP sender report packets, and because the nominal rate of the media clock is known from the payload format, the receiver can calculate the reference clock capture time for any data packet once it has received an RTCP sender report from that source. When an RTP data packet with media timestamp M is received, the corresponding reference clock capture time at the server, Ts (the RTP timestamp mapped to the reference timeline), can be calculated as follows:

$$Ts = (Ts_{sr} + (M - M_{sr})) / R$$

where, M_{sr} is the media (RTP) timestamp in the last RTCP sender report packet, Ts_{sr} is the corresponding reference clock (NTP) timestamp and R is the nominal media timestamp clock rate in hertz.

Similarly, the receiver also calculates the playout time for any particular packet, T_R according to its local reference clock. This is equal to the RTP timestamp of the packet, mapped to the receiver's reference clock timeline plus the playout buffering delay.

Once the capture and playout times are known according to the common reference timeline, the receiver can estimate the relative delay between media capture and playout for each stream. If data sampled at time Ts according to the sender's reference clock is presented at time T_R according to the receiver's reference clock, the delay between them for that given media is $D_{media} = Ts - T_R$.

Once the relative capture-to-playout delay has been estimated for different media streams, a synchronization delay between streams is computed. For example, $D_{sync} = D_{SVG} - D_{video}$. If the synchronization delay is zero, then the media streams are synchronized. A non-zero value indicates that on media is played out ahead of the other. For the media stream that is ahead, the synchronization delay (in seconds) is multiplied by the nominal media clock rate, R to convert into media timestamp units, and is then applied as a constant offset to the playout calculation for that media stream, delaying playout to match the other stream.

The choice of media for playout adjustment depends on the application, limit of human perception of error, and the priority of the media. Typically audio is more sensitive to playout adjustment when compared to other media such as SVG and video. In this case, it may be appropriate to delay SVG animations or video to match the audio presentation time.