NMOS Technical Overview

Changelog

Date	Change
2016-02-01	Initial Release

1 Introduction

The Advanced Media Workflow Association are creating a family of Networked Media Open Specifications (NMOS)

This document provide a high-level technical overview of the NMOS data model and API specifications, including examples of how these may be used.

Please check the NMOS public GitHub repository for the latest version of this document, the specifications and supporting material:

https://github.com/AMWA-TV/nmos

NOTE: at the time of writing, some supporting material is not yet available at this repository. Please contact AMWA for access to these items.

This document is non-normative.

2 Data Model Overview

Relevant specifications and supporting material for this topic

- **NMOS RTP Specification**
- **Identity, timing and sync scenarios**
- **Identifiers: lifetime and generation**

NMOS uses a logical data model based on the JTNM RA to add identity, relationships and time-based information to content and broadcast equipment. Hierarchical relationships group related entities, with each entity having its own UUID (ID and UUID are used interchangeably in this document).

The traditional assumption of 'different connectors for different signals' is replaced by the use of logical interfaces on common network interfaces, exposing video, audio and data inputs and outputs, and control parameters. This makes communication easier and helps with virtualisation of broadcast equipment, with multiple devices operating on a shared physical host.

Data modelling -- identifying the important parts of a system and the relationships between them -- has an essential role to play in specifying what goes over the logical interfaces and will be an essential part of the transition of broadcasting and media production infrastructure to make the most of IP.

3 Node

Nodes are logical hosts for processing and network operations. They may have a permanent physical presence, or may be created on demand, for example as a virtual machine in a cluster or cloud. Connections between Nodes through which content is transported are created to build a functioning broadcast plant.

3.1 Devices

Nodes provide **Devices**, which are logical groupings of functionality (such as processing capability, network inputs and outputs).

3.2 Sources, Flows and Grains

Devices with the capability to originate content must provide an abstract logical point of origin for this, called a **Source**. A Device may have multiple Sources. Sources are the origin of one or more **Flows**, which are concrete representations of content.

Flows are composed of sequences of **Grains**. A Grain represents an element of Essence or other data associated with a specific time, such as a frame, a group of consecutive audio samples, or captions.

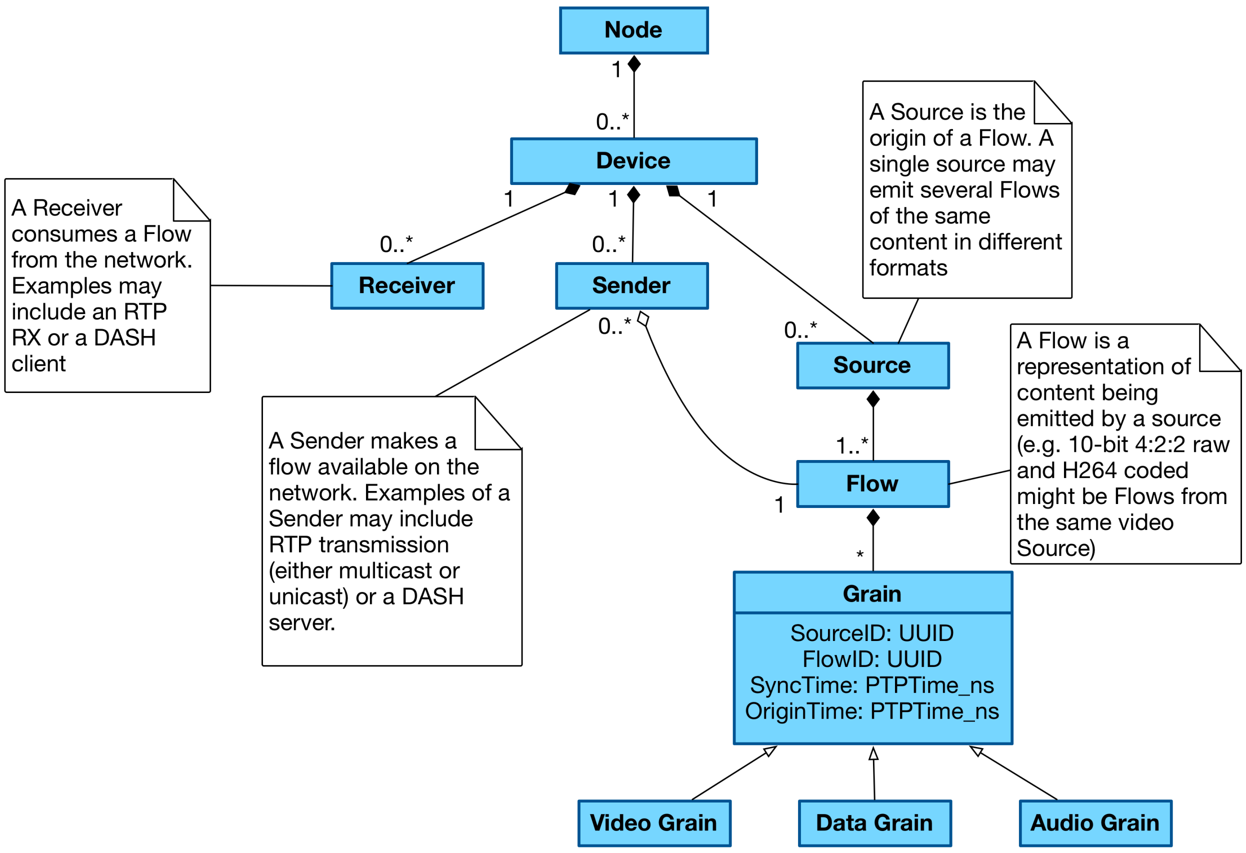
Grains also contain metadata information that specifies attributes such as the temporal duration of the payload, useful timestamps, originating Source ID and the Flow ID the grain is associated with.

As an example, consider the above concepts in the context of a video camera with an on-board microphone:

- The camera itself is a Node, which provides a single Device with two Sources (one audio Source, one video Source).
- The audio Source provides a single audio Flow.
- The video Source provides (for the purposes of this example) two video Flows, one uncompressed, the other mezzanine encoded.

3.3 Senders and Receivers

Devices transmit and receive Flows over the network using **Senders** and **Receivers.** These can be respectively considered as "virtual output ports" and "virtual input ports" on the Device. Receivers connect to Senders in order to transport content.


Node
1
0..*
Device
A Receiver consumes a Flow from the network. Examples may include an RTP RX or a DASH client
1
1
1
0..*
0..*
Receiver
Sender
A Source is the origin of a Flow. A single source may emit several Flows of the same content in different formats
0..*
0..*
Source
A Sender makes a flow available on the network. Examples of a Sender may include RTP transmission (either multicast or unicast) or a DASH server.
1..*
Flow
1
A Flow is a representation of content being emitted by a source (e.g. 10-bit 4:2:2 raw and H264 coded might be Flows from the same video Source)
*
Grain
SourceID: UUID
FlowID: UUID
SyncTime: PTPTime_ns
OriginTime: PTPTime_ns
Video Grain
Data Grain
Audio Grain

4 Accessing a Node's resources

Relevant specifications and supporting material for this topic
• **NMOS Node API Specification**

Each Node represents the resources in its data model using a RESTful HTTP API, the structure of which is described in the *Node API specification* document.

5 Identification, synchronisation and transport of Flows

Relevant specifications and supporting material for this topic
• **NMOS RTP Specification** • **Timing and synchronisation**

Each Grain contains Flow and Source UUIDs as well as other attributes such as an **Origin Timestamp** and **Synchronisation Timestamp** (these are typically identical, but may differ in some cases, consult the *Timing and synchronisation* accompanying document for further information on timestamps and their uses).

How this data travels between Nodes depends on the transport protocol being used. For each transport, the NMOS data model must be mapped into appropriate places in the protocol.

As an example, RTP is typically used for real-time transport. RTP provides customisable *header extensions* into which the NMOS data model fields are inserted. For further information consult the *NMOS RTP specification*..

6 Timing Model

Relevant specifications and supporting material for this topic

- **Identity, timing and sync scenarios**
- **Timing and synchronisation**

The NMOS Timing model has been designed to be used with a clock derived from PTP in order to synchronise Devices within a facility. By pairing this clock with a GPS source, absolute timing accuracy between geographically distributed locations can be ensured. As per SMPTE ST 2059-1, an epoch of 1970-01-01T00:00:00TAI is used, with zero phase at the epoch for periodic signals.

The capture of essence such as video frames is governed by this clock, providing frame synchronisation between discrete Devices. Each Grain is timestamped using the PTP clock in order to provide the means for synchronisation of Flows where required, and unique identity for Grains in time. Two timestamps are used for each Grain:

- **Origin Timestamps** provide the original sampling instant of the media at the edge of the system, uniquely referenceable for all time.

- **Synchronisation Timestamps** provide a means to cross-reference between Flows which may have passed through the network fabric via different paths, or passed through processing chains (such as codecs) which impose different levels of delay.

By pairing these timestamps with Flow identifiers, it is possible to track a video frame or other essence through its chain of ancestors right back to the Device which originally captured it and the instant in time when that occurred.

Consult the *Timing and synchronisation* accompanying document for further information on this topic.

7 Example System Architecture

This section describes the overarching structure of a broadcast plant using NMOS data model and APIs. It explores how Nodes communicate, and how a multi-Node system is created.

7.1 Node Structure

Relevant specifications and supporting material for this topic
• **Node API Specification**

Regardless of their implementation, viewed logically, Nodes provide:

- An HTTP API to allow clients to view and manipulate the Node data model.
- Interfaces (in the logical sense) through which content is transported.
- A PTP slave for timing and synchronisation.

More detail on the Node HTTP API is contained in the *Node API specification.*

NMOS does not specify the internal interfaces within a Node.

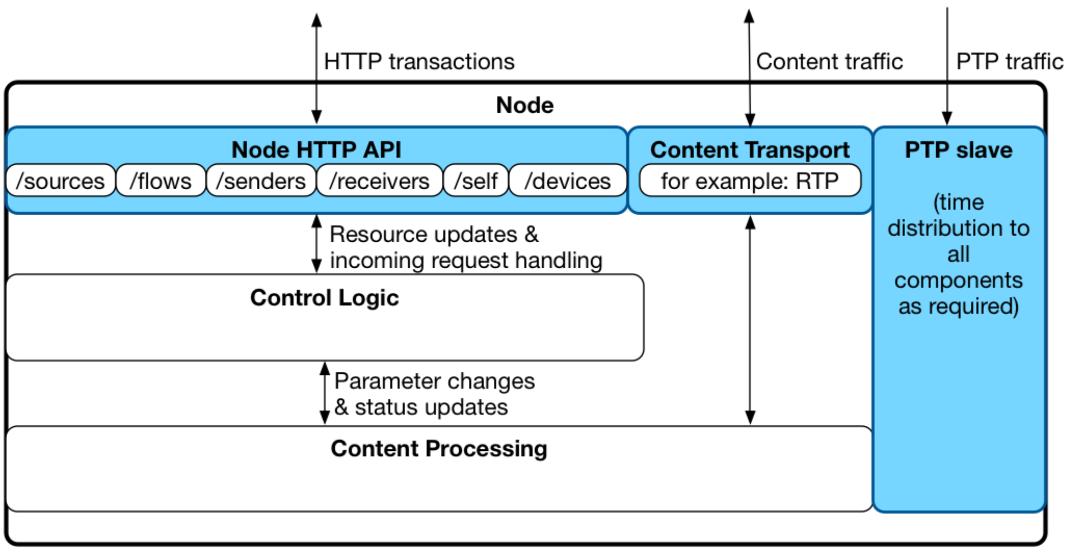


Figure 2 - Node Components (Logical View). Highlighted components are required by the Incubator specification.

7.2 Registering and Discovering Nodes

Relevant specifications and supporting material for this topic

- **Node API specification**
- **Registration API specification**
- **Query API specification**
- **Peer-to-Peer discovery**

Node and resource discovery is an important component of the NMOS family of specifications. There are two ways in which this can take place, peer-to-peer and 'registered'. Note that these two mechanisms may co-exist if this is operationally useful.

7.2.1 Peer-to-Peer Discovery

Peer-to-peer (P2P) discovery requires no additional infrastructure. Nodes make DNS Service Discovery (DNS-SD) announcements regarding the presence of their Node API. Peers browse for appropriate DNS records and then query the node HTTP API for further information.

For further information read *Peer-to-Peer discovery*.

7.2.2 Registered Model

Registered discovery takes places using a Registration & Discovery System (RDS), which is designed to be modular and distributed. An RDS is composed of one or more Registry & Discovery Instances (RDIs). Each RDI provides three services:

- A **Registration Service**
- A **Query Service**
- A **Registry** storage backend.

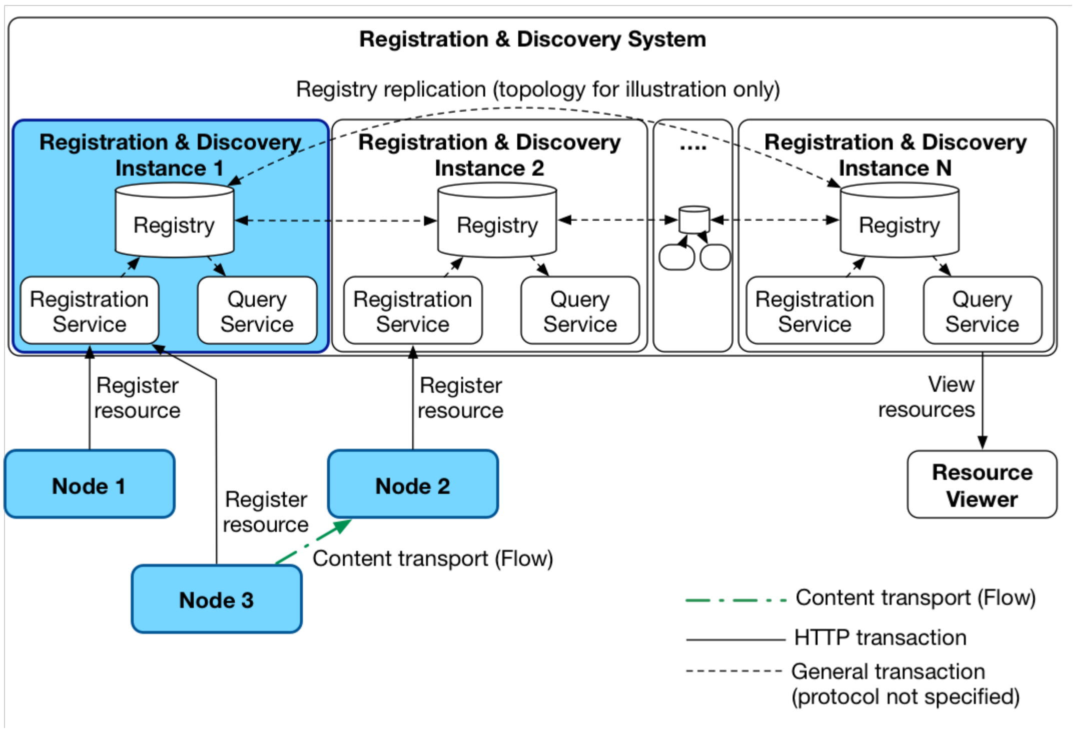


Figure 3 - Nodes and the Registration & Discovery System.

The Registration Service provides a HTTP endpoint which Nodes POST into to register themselves and their resources. The Registration Service also manages garbage collection of Nodes and their resources by requiring Nodes to send regular keep-alive/heartbeat messages.

The Query Service provides a HTTP endpoint from which clients can GET lists of resources. Typical usage examples include:

- Obtaining a list of registered Nodes in order to drive a configuration interface.
- Obtaining a list of Sender resources and a list of Receiver resources in order to provide a connection management interface.

The Query API also provides the ability to generate 'long lived' queries using its Subscription mechanism and WebSockets. This is further described in the *NMOS Query API Specification*.

More detail on registered discovery is provided in the *Registration*, *Query* and *Node API* specification documents.

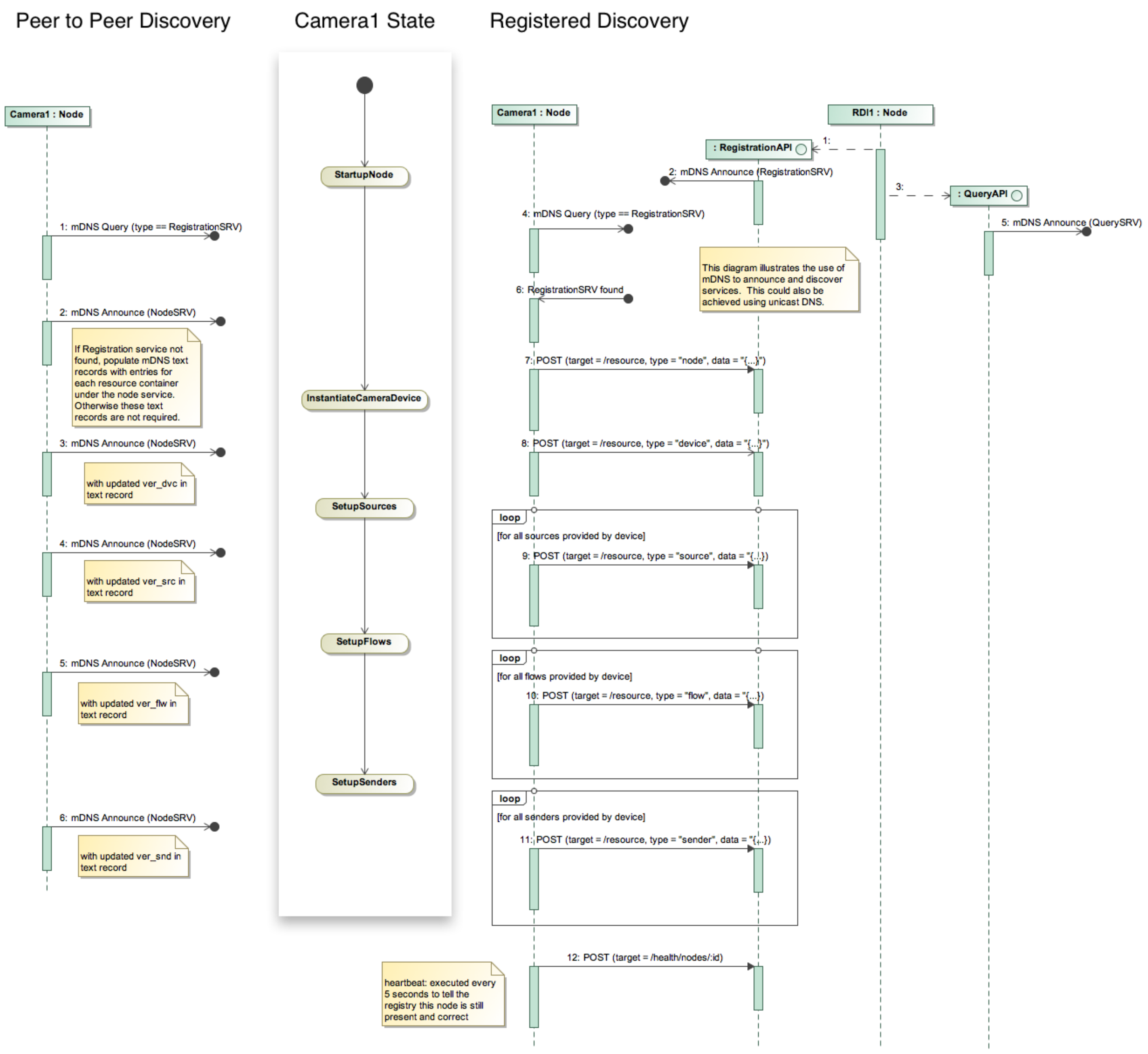

Peer to Peer Discovery
Camera1 State
Registered Discovery
Camera1 : Node
1: mDNS Query (type == RegistrationSRV)
2: mDNS Announce (NodeSRV)
If Registration service not found, populate mDNS text records with entries for each resource container under the node service. Otherwise these text records are not required.
3: mDNS Announce (NodeSRV)
with updated ver_dvc in text record
4: mDNS Announce (NodeSRV)
with updated ver_src in text record
5: mDNS Announce (NodeSRV)
with updated ver_flw in text record
6: mDNS Announce (NodeSRV)
with updated ver_snd in text record
StartupNode
InstantiateCameraDevice
SetupSources
SetupFlows
SetupSenders
Camera1 : Node
RDI1 : Node
: RegistrationAPI
1:
2: mDNS Announce (RegistrationSRV)
3:
: QueryAPI
5: mDNS Announce (QuerySRV)
4: mDNS Query (type == RegistrationSRV)
This diagram illustrates the use of mDNS to announce and discover services. This could also be achieved using unicast DNS.
6: RegistrationSRV found
7: POST (target = /resource, type = "node", data = "{...}")
8: POST (target = /resource, type = "device", data = "{...}")
loop
[for all sources provided by device]
9: POST (target = /resource, type = "source", data = "{...})
loop
[for all flows provided by device]
10: POST (target = /resource, type = "flow", data = "{...})
loop
[for all senders provided by device]
11: POST (target = /resource, type = "sender", data = "{...})
12: POST (target = /health/nodes/:id)
heartbeat: executed every 5 seconds to tell the registry this node is still present and correct

Figure 4 - Registration Sequence and State Diagrams (peer-to-peer and registered mechanisms are both illustrated)

8 Connection Management

Relevant specifications and supporting material for this topic

- **Example SDP file for RFC4175 video stream**
- **Example SDP file for L24 audio stream**
- **Example SDP file for ST.291 ANC stream**

Sender and Receiver resources enable connectivity between Nodes. Senders expose a description of what Flow they are sending and how to access it (the 'manifest href').

For RTP, we use SDP files as our 'manifest'. Senders expose an SDP file that describes the location (and type) of the Sender's RTP stream.

To form a connection between two Nodes, a connection manager PUTs some structured data about a Sender (including a reference to the manifest href) to a Receiver. The Receiver parses the manifest and begins accessing the stream.

Each time a Receiver changes the Sender it is receiving from, the Node resource describing the Receiver must be updated on the Node, and also the Node must notify the Registration & Discovery system.

Further information on this topic is available in the *NMOS Node API Specification*, and the example SDP files. Figure 5 also illustrates a typical connection management sequence.

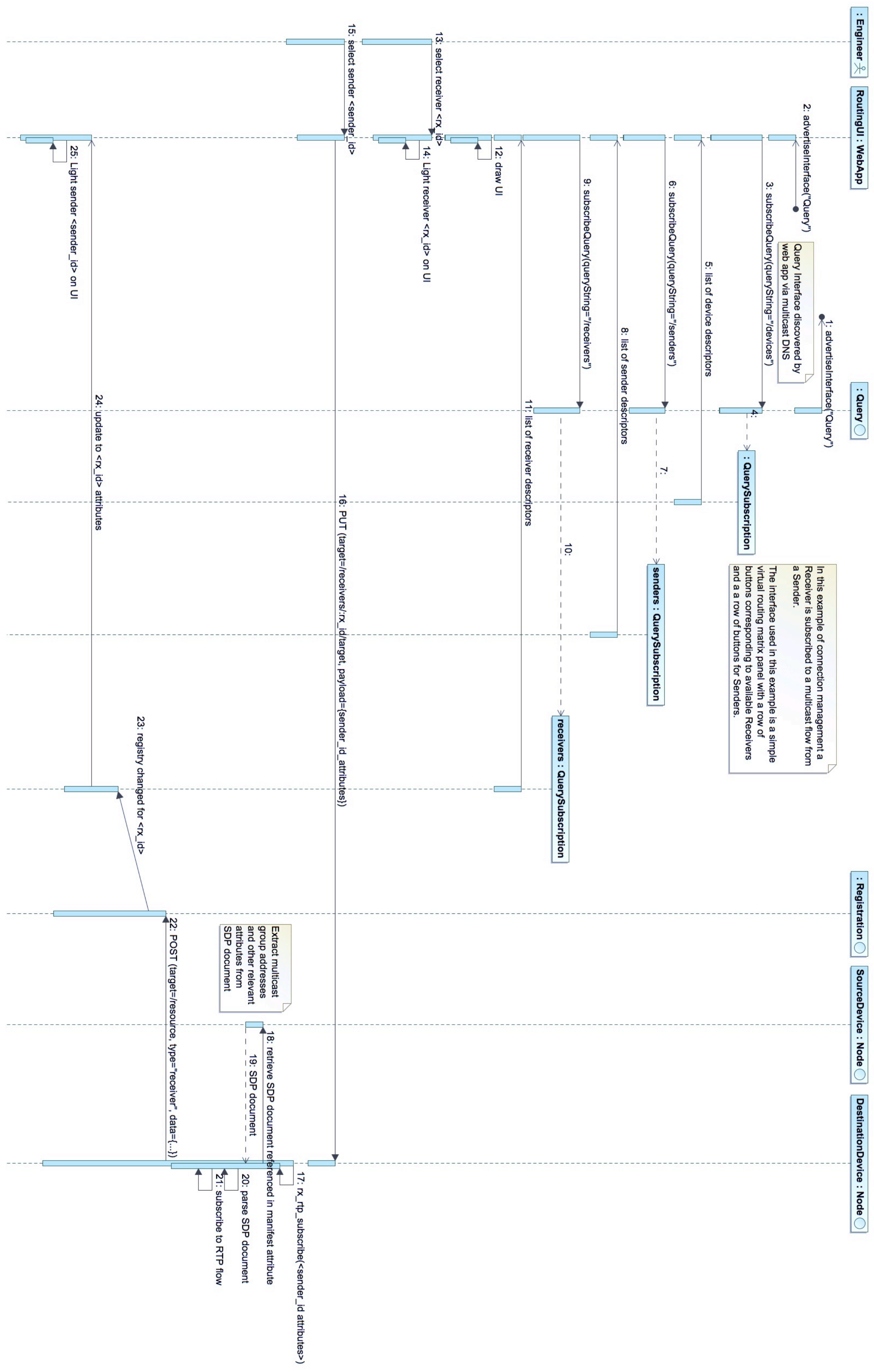

: Engineer
RoutingUI : WebApp
: Query
: QuerySubscription
senders : QuerySubscription
receivers : QuerySubscription
: Registration
SourceDevice : Node
DestinationDevice : Node
1: advertiseInterface("Query")
2: advertiseInterface("Query")
Query Interface discovered by web app via multicast DNS
3: subscribeQuery(queryString="/devices")
4:
5: list of device descriptors
6: subscribeQuery(queryString="/senders")
7:
8: list of sender descriptors
9: subscribeQuery(queryString="/receivers")
10:
11: list of receiver descriptors
12: draw UI
13: select receiver <rx_id>
14: Light receiver <rx_id> on UI
15: select sender <sender_id>
16: PUT (target=/receivers/:rx_id/target, payload={sender_id_attributes})
17: rx_rtp_subscribe(<sender_id attributes>)
18: retrieve SDP document referenced in manifest attribute
19: SDP document
20: parse SDP document
21: subscribe to RTP flow
22: POST (target=/resource, type="receiver", data={...})
23: registry changed for <rx_id>
24: update to <rx_id> attributes
25: Light sender <sender_id> on UI
In this example of connection management a Receiver is subscribed to a multicast flow from a Sender.
The interface used in this example is a simple virtual routing matrix panel with a row of buttons corresponding to available Receivers and a a row of buttons for Senders.
Extract multicast group addresses and other relevant attributes from SDP document

Figure 5 - Connection Management Sequence Diagram

9 Content Transport

Relevant specifications and supporting material for this topic

- **NMOS RTP specification**
- **RFC 4175 RTP stream packet capture (with NMOS header extensions)**
- **L24 audio RTP stream packet capture (with NMOS header extensions)**
- **ST.291 ANC RTP stream packet capture (with NMOS header extensions)**

NMOS's content model can be applied to several types of payload format and transport protocol, including, but not limited to:

- Uncompressed video over RTP according to RFC 4175
- Uncompressed audio over RTP according to RFC 3190
- ST.291 ANC data over RTP according to draft-ietf-payload-rtp-ancillary-02

NMOS RTP Specification describes how the above mappings, including RTP header extensions to carry identity and timing information and signal Grain boundaries. These can be extended to support e.g. mezzanine compression formats such as VC-2.