CLUE Working Group R. Presta
Internet-Draft S P. Romano
Intended status: Standards Track University of Napoli
Expires: May 9, 2014 November 5, 2013

CLUE protocol
draft-presta-clue-protocol-03

Abstract

The CLUE protocol is an application protocol conceived for the description and negotiation of a CLUE telepresence session. The design of the CLUE protocol takes into account the requirements and the framework defined, respectively, in [I-D.ietf-clue-framework] and [I-D.ietf-clue-telepresence-requirements]. The companion document [I-D.kyzivat-clue-signaling] delves into CLUE signaling details, as well as into the SIP/SDP session establishment phase. We herein focus on the application level perspective. Message details, together with the behavior of CLUE participants acting as Media Providers and/or Media Consumers, are discussed.

Table of Contents

1. Introduction

The CLUE protocol is an application protocol used by two CLUE Participants to enhance the experience of a multimedia telepresence session. The main goals of the CLUE protocol are:

1. enabling a MP to advertise its current telepresence capabilities to the MC in terms of available media captures, encodings, and simultaneity constraints;

2. enabling a MC to request the desired multimedia streams from the offering MP.

CLUE Participants are connected by means of the CLUE signaling channel. Such channel has been conceived as a DTLS/SCTP/UDP channel in [I-D.kyzivat-clue-signaling] and it is established as depicted in the same document. CLUE protocol messages flow across such channel.

While [I-D.kyzivat-clue-signaling] focuses on protocol signaling details and on its interaction with the SIP/SDP session establishment phase, we herein investigate the protocol in action. We assume the DTLS/SCTP/UDP channel is established and define the behavior of the CLUE Participants communicating on it. We discuss how the CLUE dialogue between them can be exploited to successfully setup the telepresence session according to the principles and concepts pointed out in in [I-D.ietf-clue-framework].

In Section 3 we provide an overview of the CLUE protocol and describe CLUE messages along with their features and functionality. The CLUE participant state machine is introduced in Section 4. Versioning, extensions and options management mechanisms are discussed in Section 8, Section 9 and Section 10, respectively. The XML schema defining the CLUE messages is reported in Section 11.

2. Terminology

This document refers to the same terminology used in [I-D.ietf-clue-framework] and in [I-D.ietf-clue-telepresence-requirements]. We briefly recall herein some of the main terms used in the document. We further introduce the definition of CLUE participant.

CLUE Participant An entity able to use the CLUE protocol within a telepresence session. It can be either an endpoint or an MCU able to use the CLUE protocol.

Endpoint The logical point of final termination through receiving, decoding and rendering, and/or initiation through capturing, encoding, and sending of media streams. An endpoint consists of one or more physical devices which source and sink media streams, and exactly one [RFC4353] Participant (which, in turn, includes exactly one SIP User Agent). Endpoints can be anything from multiscreen/multicamera room controllers to handheld devices.

MCU Multipoint Control Unit (MCU) - a device that connects two or more endpoints together into one single multimedia conference [RFC5117]. An MCU may include a Mixer [RFC4353].

Media Any data that, after suitable encoding, can be conveyed over RTP, including audio, video or timed text.

Media Capture A "Media Capture", or simply "Capture", is a source of Media.

Capture Encoding A specific encoding of a Media Capture, to be sent via RTP [RFC3550].

Media Stream The term "Media Stream", or simply "Stream", is used as a synonymous of Capture Encoding.

Media Provider A CLUE participant (i.e., an Endpoint or an MCU) capable to send Media Streams.

Media Consumer A CLUE participant (i.e., an Endpoint or an MCU) capable to receive Media Streams.

3. Overview of the CLUE protocol

The CLUE protocol has been conceived to enable telepresence sessions. It is designed in order to address SDP limitations in terms of the description of several qualitative parameters about the multimedia streams that are involved in a real-time multimedia conference session. Indeed, by simply using SDP we are not able to convey all the information about the features of the flowing multimedia streams that is needed to enable a "being there" rendering experience. Such information is being designed in the CLUE framework draft and formally defined and described in the CLUE data model draft. The CLUE protocol represents the mechanism that enables the exchange of CLUE information between CLUE participants.

The CLUE protocol, as defined in this document, is a stateful, client-server, XML-based application protocol. CLUE protocol messages flow on a DTLS/SCTP/UDP channel connecting two CLUE Participants. The main goals of the CLUE protocol are:

1. enabling a MP to advertise its current telepresence capabilities to the MC in terms of available media captures, encodings, and simultaneity constraints;

2. enabling a MC to request the desired multimedia streams from the offering MP.

Three main design layers can be identified:

1. Establishment of the CLUE channel.

2. Negotiation of the CLUE protocol version and extensions

3. Media session description and negotiation

Signaling issues about the CLUE channel establishment are considered in [I-D.kyzivat-clue-signaling]. In particular, the CLUE channel is a DTLS/SCTP/UDP channel connecting two CLUE Participants. While [I-D.kyzivat-clue-signaling] focuses on protocol signaling details and on its interaction with the SIP/SDP session establishment phase, we herein investigate the protocol in action at the CLUE application level.

As soon as the channel is ready, the CLUE Participants must agree on the protocol version and extensions to be used within the telepresence session. A mechanism for the negotiation of the CLUE protocol version and extensions is proposed in Section 10. According to such solution, the CP which is the CLUE Channel Initiator (CI) issues a proper CLUE message (OPTIONS) to the CP which is the Channel Receiver (CR). Such a message specifies the supported version and extensions. The CR answers by selecting the subset of the CI's extensions that it is able to support and determines the protocol version to be used.

After that negotiation phase is completed, CLUE Participants define the characteristics of the media streams to be exchanged in both directions. Indeed, being A and B the considered CLUE Participants, it is possible to distinguish between two dialogues:

1. the one needed to describe and set up the media streams sent from A to B, i.e., the dialogue between A's Media Provider side and B's Media Consumer side

2. the one needed to describe and set up the media streams sent from B to A, i.e., the dialogue between B's Media Provider side and A's Media Consumer side

CLUE messages for the media session description and negotiation are

designed by considering the MP side as the server side of the protocol, since it produces and provides media streams, and the MC side as the client side of the protocol, since it requests and receives media streams. The messages that are exchanged to set up the telepresence media session are described by focusing on a single MP-MC dialogue.

The MP first advertises the media captures and associated encodings to the MC, as well as possible simultaneity constraints. The description of such telepresence features is made according to the information defined in the CLUE framework and data model ([I-D.ietf-clue-framework] and [I-D.ietf-clue-data-model-schema]). The CLUE message conveing the MP's multimedia offer is the ADVERTISEMENT message. Such message leverages the XML definitions provided in [I-D.ietf-clue-data-model-schema] for the description of media captures, encodings, and simultaneity constraints features.

The MC selects the desired streams coming from the MP by using the CONFIGURE message, which makes reference to the information carried in the ADVERTISEMENT previously received by the MP.

In the following, a bird's-eye view of the CLUE protocol messages is provided. For each message it is indicated who sends it, who receives it, a brief description of the information it carries, and how/when it is used. Besides ADVERTISEMENT and CONFIGURE, new messages have been conceived in order to provide all the mechanisms and operations envisaged in [I-D.kyzivat-clue-signaling].

- ADVERTISEMENT (ADV)
- ACKNOWLEDGEMENT (ACK)
- CONFIGURE (CONF)
- CONFIGURE RESPONSE
- RE-ADV
- RE-ADV RESPONSE
- OPTIONS
- OPTIONS RESPONSE

3.1. ADVERTISEMENT

FROM	MP
TO	MC
TYPE	Notification
DESCRIPTION	This message is used by the MP to advertise the available media captures and related information to the MC. The ADV contains elements compliant with the CLUE data model and other information like the CLUE protocol version and a sequence number.
USAGE	The MP sends this message as soon as the CLUE channel is ready. The MP sends an ADV to the MC each time there is a modification of the MP's telepresence capabilities. The ADV message is also sent back to the MC when the MP receives a RE-ADV request.

The ADV message is considered a notification since, during the session, it can be sent from the MP also on a per-event basis, i.e. when the CLUE capabilities of the MP change with respect to the last issued ADV. It is still to be discussed if a "delta" mechanism for advertising only the changes with respect to the previous notification should be adopted. Similar approaches have been proposed for partial notifications in centralized conferencing frameworks ([RFC6502]), leveraging the XML diff codification mechanism defined in [RFC5261].

3.2. CONFIGURE

FROM	MC
TO	MP
TYPE	Request
DESCRIPTION	This message allows a MC to ask for the desired (advertised) capture. It contains capture encodings and other information like the CLUE protocol version and a sequence number.
USAGE	The MC can send a CONF after the reception of an ADV or each time it wants to request other advertised captures from the MP.

3.3. RESPONSE

FROM	MP
TO	MC
TYPE	Response
DESCRIPTION	This message allows a MP to answer a CONF message. Besides the protocol version and a sequence number, it contains a response code with a response string indicating either the success or the failure (along with failure details) of a CONF request elaboration. Example response codes and strings are provided in the following table.
USAGE	The MP sends this message in response to a CONF message.

Response codes can be designed by adhering to the HTTP semantics, as shown below.

Response code	Response string	Description
410	Bad syntax	The XML syntax of the CONF message is not correct.
411	Invalid value	The CONF message contains an invalid parameter value.
412	Invalid identifier	The identifier used for requesting a capture is not valid or unknown.
413	Conflicting values	The CONF message contains values that cannot be used together.
420	Invalid sequencing	The sequence number of the CONF message is out of date or corresponds to an obsoleted ADV.
510	Version not supported	The CLUE protocol version of the CONF message is not supported by the MP.
511	Option not supported	The option requested in the CONF message is not supported by the MP.

... TBC.

Response code family	Description
1XX	Temporary info
2XX	Success
3XX	Redirection
4XX	Client error
5XX	Server error

3.4. RE-ADV

FROM	MC
TO	MP
TYPE	Request
DESCRIPTION	This message allows a MC to request a MP to issue a new copy of the ADV. This message can contain a reason string indicating the motivation for the request (e.g., refresh, missing elements in the received ADV, wrong syntax in the received ADV, invalid capture area, invalid line of capture point, etc).
USAGE	The MC sends this message to the MP when the timeout for the ADV is fired, or when the ADV is not compliant with the CLUE specifications (this can be useful for interoperability testing purposes)

3.5. OPTIONS

ToDo. See Section 10.

4. Protocol state machines

The CLUE protocol is an application protocol used between a Media Provider (MP) and a Media Consumer (MC) in order to establish a multimedia telepresence session. CLUE protocol messages flow upon a DTLS/SCTP channel established as depicted in [I-D.kyzivat-clue-signaling]. Over such a channel there are typically two CLUE streams between the channel terminations flowing in opposite directions. In other words, typically, both channel terminations act simultaneously as a MP and as a MC. We herein discuss the state machines associated, respectively, with the CLUE Participant, with the MC process and with the MP process.

5. CLUE Participant state machine

The main state machines focus on describing the states of CLUE channel from a CLUE channel initiator/receiver perspective. In the IDLE state, when the CP has established a CLUE channel, the main state moves to the ESTABLISHED state. When in the ESTABLISHED state, if the CP is the Channel Initiator (CI), it sends an OPTIONS message for version negotiation; otherwise, if the CP is the Channel Receiver (CR), it listens to the channel for an OPTIONS message associated with version negotiation. If an OPTIONS message is sent (or received), the CP moves to the NEGOTIATING state. If the CP detects errors in the request message received, the main state goes back to the IDLE state. When in the NEGOTIATING state, the CR prepares an OPTIONS response message while the CI listens to the channel for an OPTIONS response. If an OPTIONS response message for version negotiation is sent (or received), the main state moves to the ACTIVE state. If the CP detects errors in the response message received or receives an error response, it goes back to the IDLE state. When the party enters the ACTIVE state, it creates two sub state machines: the MC state machine and the MP state machine. When in the ACTIVE state, if the CP receives an OPTIONS message for version negotiation (or an OPTIONS response message for version negotiation), it must ignore the message and stay in the ACTIVE state. When in the ACTIVE state, a CP which receives CLUE messages (including ADV, RE-ADV and CONF, as well as the corresponding response messages) dispatches them to the proper underlying sub state machine for further processing. The same happens in case of changes in the telepresence settings. The TERMINATED state is reachable from each of the aforementioned states whenever the session is canceled or released. The IDLE state is reachable from each of the aforementioned states whenever the underlying channel is closed (only due to connection errors).

```
                               +-------------+<------------------------------
+-------------+
                  +------------>+     IDLE    +<--------------------------+
+   TIME OUT  +
                  |            +------+------+<-----------+              |
+------+------+
                  |                   |                   |
|          |
                  |                   |                   |
|          |
              Connection             CLUE                 |
|          |
                error               channel               |
|          |
                  |            has been established       |
|          |
                  |                   |                   |
|          |
                  |                   V               Receive error
|          |
                  +------------+-------------+     (version mismatch,
|          |
            +------------------+ ESTABLISHED +      missing elements,..)
|       time out
            |                  +------+------+            |
|          |
            |                         |                   |
Connection     |
            |                         |                   |
error       |
            |                  Send/Receive request       |
|          |
            |                  Version negotiation        |
|          |
            |                         |                   |
|          |
            |                         V                   |
|          |
            |                  +-------------+------------+
|          |
            |     +------------+ NEGOTIATING +----------------------------
+          |
            |     |            +------+------
+--------------------------|----------+
            |     |                   |                                |
            |     |                   |                                |
            |     |          Receive/Send response                Connection
            |     |           Version negotiation                    error
```

```
         Session  |                   |                                  |
           end    |                   V                                  |
            |     |            +-------------+---------------------------+
            |     |            |   ACTIVE    +<-------------------+
            |     |            |  +-------+  |           Receive(request/
response)
            |     |            |  |SUBIDLE|  |             Version negotiation
            |   Session        |  |MC     |  +--------------------+
            |    end           |  +-------+  |
            |     |            |  +-------+  +<-------------------+
            |     |            |  |SUBIDLE|  |       Send/Receive(ADV/CONF/RE-
ADV/RESP)
            |     |            |  |MP     |  |         Change telepresence
settings
            |     |            |  +-------+  |                    |
            |     |            +------+------+--------------------+
            |     |                   |
            |     |                   |
            |     |                Session
            |     |                  end
            |     |                   |
            |     |                   V
            |     |            +-------------+
            |     +----------->+ TERMINATED  +
            +----------------->+-------------+
```

6. Media Consumer's state machine

An MC in the WAIT FOR ADV state is waiting for an ADV coming from the MP. If the timeout expires ("timeout"), the MC switches to the TIMEOUT state.

In the TIMEOUT state, if the number of trials is below the retry threshold, the MC sends a RE-ADV/refresh message to the MP ("send RE-ADV"), switching back to the WAIT FOR ADV. Otherwise, the MC moves to the TERMINATED state.

When the ADV has been received ("receive ADV"), the MC goes into the ADV RECEIVED state. The ADV is then parsed. If something goes wrong with the ADV (bad syntax, missing XML elements, etc.), the MC sends a NACK message to the MP specifying the encountered problem via a proper reason phrase. In this way, the MC switches back to the WAIT FOR ADV state, waiting for a new copy of the ADV. If the ADV is successfully processed, the MC issues an ACK message to the MP and moves to the ADV ACKED state. When the CONF request is ready, the MC sends it and moves to the TRYING state. Alternatively, if the ADV is successfully processed, and the CONF request is timely available, the MC can piggyback the ACK message within a CONF request and move from the ADV RECEIVED state directly to the TRYING state.

While in the TRYING state, the MC is waiting for a CONF RESPONSE message (to the issued CONF) from the MP. If the timeout expires ("timeout"), the MC moves to the TIMEOUT state and sends a RE-ADV in order to solicit a new ADV from the MP. If a CONF RESPONSE with an error code is received ("receive 4xx, 5xx not supported"), then the MC moves back to the ADV RECEIVED state and produces a new CONF message to be sent to the MP. If a successful RESPONSE arrives ("receive 200 OK"), the MC gets into the CONF COMPLETED state. state.

When the MC is in the CONF COMPLETED state, it means that the telepresence session configuration has been set up according to the MC's preferences. Both the MP and the MC have agreed on (and are aware of) the media streams to be exchanged within the call. If the MC decides to change something in the call settings, it issues a new CONF ("send CONF") and moves back to the TRYING state. If a new ADV arrives from the MP ("receive ADV"), it means that something has changed on the MP's side. The MC then moves to the ADV-RCV state and prepares a new CONF taking into account the received updates. When the underlying channel is closed, the MC moves into the TERMINATED state.

The TERMINATED state is reachable from each of the aforementioned states whenever the underlying channel is closed. The corresponding transitions have not been reported for the sake of simplicity. This

termination condition is a temporary solution.

```
                                                                      +-----+
                     +---------------+-------------timeout------>+----+--+  |
                     | WAIT FOR ADV  |<----+                     |TIMEOUT|  |
                     +---------------+<----+--------send---------+-------+  |
                                |          |     RE-ADV(refresh)      ^     |
                                |          |                          |     |
                                |          |                          |     |
                             receive     send                         |     |
                               ADV       NACK                         |     |
           +---receive-------+  |       (missing elements,            |     |
           |   error RESP    |  |        invalid area,...)            |     |
           |                 v  v          |                          |     |
           +----------------+---------+----+      +--------+          |     |
          +---------------->|  ADV    |---send--->|  ADV   |      timeout   |
          |                 | RECEIVED|   ACK     | ACKED  |          |     |
          |     +---------->|         |           |        |          |     |
          |    recv  +----->+-----+---+<--recv----+----+---+          |     |
          |   error  |            |       ADV          |              |     |
          |   RESP   |          send                   |              |     |
      receive   +    |          CONF,              send|              |     |
        ADV     |    |         CONF+ACK            CONF|              |     |
          |     |    |            |                    |              |     |
          |     |  receive        v                    |              |     |
          |     |  ADV         +-----------+<----------+              |     |
          |     |    |         |           |+-------------------------+     |
          |     +----|--------+|  TRYING   |                                |
          +----------|---------|           |                                |
                 +---|---------+-----------+                                |
                 |   |          |          ^                                |
                 |   |          |          |                                |
                 |   |          |          |                                |
          receive|   |         receive    send                          retry
          error RESP,|          200 OK    CONF                        expires
          retry  |   |          |          |                                |
          expired|   |          |          |                                |
                 |   |          |          |                                |
                 |   |          v          |                                |
                 |   |       +---------+   |                                |
                 |   +-------| CONF    |   |                                |
                 |           |COMPLETED|---+                                |
                 |           +---------+                                    |
                 |              |                                           |
                 |              |                                           |
```

```
                 |              |                                           |
                 |              |                                           |
                 |            connection                                    |
                 |             closed                                       |
                 |              |                                           |
                 |              |                                           |
                 |              |                                           |
                 |              |                                           |
                 |              |                                           |
                 |              v                                           |
                 |            +----------+<---------------------------------+
                 +----------->+TERMINATED|
                              +----------+
```

7. Media Provider's state machine

In the PREPARING ADV state, the MP is preparing the ADV message reflecting the actual telepresence capabilities. After the ADV has been sent, the MP moves to the WAIT FOR ACK state. If the ACK arrives, the MP moves to the WAIT FOR CONF state. If a NACK arrives, it goes back to the PREPARING ADV state.

When in the WAIT FOR ACK state, if a CONF or a CONF+ACK arrives, the MP switch to the CONF RECEIVED state directly.

When in the WAIT FOR CONF state, the MP is listening to the channel for a CONF coming from the MC. If a RE-ADV is received, the MP goes back to the IDLE state and issues an ADV again. If telepresence settings change in the meanwhile, it moves back to the PREPARING ADV state and prepares a new ADV to be sent to the MC. If a CONF arrives, the MP switches to the CONF RECEIVED state. If nothing happens and the timeout expires, than the MC falls into the TIMEOUT state.

In the TIMEOUT state, if the number of trials does not exceed the retry threshold, the MC comes back to the PREPARING ADV state for sending a new ADV. Otherwise, it goes to the TERMINATED state.

The MP in the CONF RECEIVED state is processing the received CONF in order to produce a CONF RESPONSE message. If the MP is fine with the MC's configuration, then it sends back a 200 OK successful CONF RESPONSE and moves to the IN CALL state. If there are errors duting CONF processing, then the MC returns a CONF RESPONSE carrying an error response code. Finally, if there are changes in the telepresence settings, it goes back to the PREPARING ADV state to issue an updated ADV.

When in the CONF COMPLETED state, the MP has successfully configured the telepresence session according to the MC's specifications. If a new CONF arrives, it switches to the CONF RECEIVED state to analyze the new request. If a RE-ADV arrives, or some modifications are applied to the telepresence options, then it moves to the PREPARE-ADV state to issue the ADV. When the channel is terminated, the MP falls into the TERMINATED state.

The TERMINATED state is reachable from each of the aforementioned states whenever the underlying channel is closed. The corresponding transitions have not been reported for the sake of simplicity. This termination condition is a temporary solution.

```
                    +-----------+
                    |           |
                    | PREPARING |
 +----------------->|   ADV     |<--------------------------+
 |   +------------->|           |<-----------retry----------+---------+
 |   |       +----->|           |<--+         not            |         |
 |   |       |      +-----------+   |        expired        |         |
 |   |       |            |         |                       |         |
 |   |  change          send       receive                 |        ++------+
 |   |  telepresence     ADV        NACK                    |        |TIMEOUT|
 |   |  settings          |         |                       |        ++--+---+
 |   |       |            |         |                       |         ^  |
 |   |       |            v         |                       |         |  |
 |   |       |    +-------------+---+                       |         |  |
 |   |       +----+  WAIT FOR   +------------timeout--------+---------+  |
 |   |         +--+     ACK     |                           |            |
 change        |  +-------+-----+                           |            |
 telepresence  |          |                                 |            |
 settings      |        recv                                |            +
 +   +         |        ACK                                 +            |
 |   |         |          |                                 |            |
 |   |         |          v                                 |            |
 |   |         |     +-----------+                          |            |
 |   |      recv     | WAIT FOR  |                          |            |
 |   |       CONF,   |   CONF    |<-+                       |            |
 |   |    CONF+ACK   +----+------+  |                       |            |
 |   |         |          |         +                       |            |
 +   |         |        receive    CONF error,              +            |
 |   +         |         CONF      retry not expired,       |            +
 |   |         |          |        send error RESP          |            |
 |   |         |          |         |                       |            |
```

```
 |   |         |          |         |                       |            |
 |   |         |          v         |                       |            |
 |   |         +--->+-----------+---+                       |            |
 +---+-------------+|   CONF    |                           |            |
     |       +----->| RECEIVED  |----CONF error,            |            |
     |       |      +-----+-----+    retry expired          |            |
     |       |            |          +                      |            |
     |       |            |          |                      |            |
     |       |            |          |                      |            |
 receive   receive      send         |                      |            |
 RE-ADV     CONF       200 OK        |                      |       retry|
     |       |            |          |                      |      expired
     |       |            |          |                      |            |
     |       |            |          |                      |            |
     |       |            v          |                      |            |
     |       |       +----------+    |   change             |            |
     |       +-------|  CONF    |----|---telepresence-------+            |
     +---------------| COMPLETED|    |   settings                        |
                     +----------+    |                                   |
                          |          |                                   |
                          |          |                                   |
                          |          |                                   |
                       connection    |                                   |
                        closed       |                                   |
                          |          |                                   |
                          |          |                                   |
                          v          |                                   +
                 +----------------+<-+                                   |
                 |   TERMINATED   |<-------------------------------------+
                 |                |
                 +----------------+
```

8. About CLUE protocol XML schema versioning

CLUE protocol messages are XML messages compliant to the CLUE protocol XML schema. The version of the protocol corresponds to the version of the schema. Both client and server have to test the compliance of the received messages with the XML schema of the CLUE protocol. If the compliance is not verified, the message cannot be processed further.

Obviously, client and server can not communicate if they do not share exactly the same XML schema. Such a schema is the one included in the yet to come RFC, and associated with the CLUE URN "urn:ietf:params:xml:ns:clue-message". If all CLUE-enabled devices use that schema there will be no interoperability problems due to schema issues.

The version of the XML schema contained in the standard document deriving from this draft will be 1.0. The subsequent versions of the XML schema should be backward compatible, not only in terms of schema but also semantically and procedurally as well. This means that they should define further features and functionality besides those defined in the previous versions, in an incremental way, without impacting the basic rules defined in the previous version of the schema. In this way, if a MP is able to speak, e.g., version 5.0 of the protocol while the MC only understands version 4.0, the MP should have no problem in reverting the dialogue to version 4.0 without exploiting 5.0 features and functionality.

It is expected that, before the CLUE protocol XML schema reaches a steady state, prototypes developed by different organizations will conduct interoperability testing. In that case, in order to interoperate, they have to be compliant to the current version of the XML schema, i.e., the one copied in the most up-to-date version of the draft defining the CLUE protocol. The versions of the non-standard XML schema will be numbered as 0.01, 0.02, and so on. During the standard development phase, the versions of the XML schema will probably not be backward compatible so it is left to prototype implementers the responsibility of keeping their products up to date.

Even though strongly discouraged, if a future version of the protocol is designed which breaks the backward compatibility constraint, this aspect MUST be explicitly advertised in the corresponding new RFC document. In such a case, it would be up to developers to update their systems accordingly.

9. Extensibility issues

Although the standard version of the CLUE protocol XML schema will be designed to thoroughly cope with the requirements emerging from the application domain, new needs might arise in the future. Such needs may relate to two main aspects of the protocol:

the information carried in the existing messages (for example, we may want to add more fields within an existing message);

the meaning of the messages. This is the case if there is no proper message for a certain task, so a brand new CLUE message needs to be defined.

9.1. Aspect 1 - new information within existing messages

CLUE messages are envelopes carrying two types of information:

XML elements defined within the CLUE protocol XML schema itself (protocol-specific information)

other XML elements compliant to the CLUE data model schema (data model information)

When new protocol-specific information is needed somewhere in the protocol messages, it can be added in place of the <any> elements and <anyAttribute> elements envisioned by the protocol schema. The policy currently defined in the protocol schema for handling <any> and <anyAttribute> elements is:

elementFormDefault="qualified"

attributeFormDefault="unqualified"

In that case, the new information must be qualified by namespaces other than "urn:ietf:params:xml:ns:clue-message" (the protocol URN) and "urn:ietf:params:xml:ns:clue-info" (the data model URN). Elements or attributes from unknown namespaces MUST be ignored.

The other matter concerns data model information. Data model information is defined by the XML schema associated with the URN "urn:ietf:params:xml:ns:clue-info". Also for the XML elements defined in such a schema there are extensibility issues. Those issues are overcome by using <any> and <anyAttribute> placeholders. Similarly to what said before, new information within data model elements can be added in place of <any> and <anyAttribute> schema elements, as long as they are properly namespace qualified.

9.2. Aspect 2 - new messages

New CLUE protocol messages, not envisioned in the standard version of the schema, are needed. Also in that case we have three chances:

writing down a new version of the protocol schema, with the new messages added after the existing ones. The same considerations of the first option above hold here.

putting all the new messages inside a brand new schema to be linked to a new URN that the most up to date telepresence system must be aware of.

designing a wildcard envelope for future messages. This is an approach used also within the CCMP protocol (Centralized Conferencing Manipulation Protocol, [RFC6503]). In that case, a mechanism for the extension negotiation is also envisioned.

10. Managing protocol version negotiation and extensions: the OPTIONS request

In this section we provide a mechanism for handling both protocol extension and version negotiation issues.

We propose a new request message issued by the CI towards the CR as soon as the CLUE channel is istantiated: the OPTIONS message. This message carries:

the CLUE protocol version adopted by the CI

the data model extensions supported by the CI

the protocol extensions supported by the CI

When the CR receives the OPTIONS message, it reads the CLUE protocol version of the CI (the highest protocol version of the CI). If the CI's version is higher than the CR's one, then the CR responds to the CI by using in the OPTIONS RESPONSE message its own version. The CI has to downgrade the CLUE dialogue to the version specified by the CR in the subsequent CLUE messages. If CI's version is equal to or lower than CR's version, then the CR will use in the OPTIONS RESPONSE message the same version as the one in the OPTIONS message and all subsequent CLUE messages must carry that version number. In the latter case, it is the CR who has to to downgrade the CLUE dialogue in order to be understood by the CI.

A data model extension is a set of XML definitions related to the description of telepresence capabilities that is contained in an XML schema and which is different from the normative CLUE data model schema. Such XML definitions can represent further entities not envisioned in the CLUE framework at the time of writing of the data model draft. The entities defined in a data model extension can appear in place of the <any> and <anyAttribute> elements included in the data model document. A data model extension is then represented by a reference to the defining XML schema. The schema reference is represented by a URI defining the schema location. [TBC] If a data model extension is supported by both a CR and a CI, this means that both are aware of the associated XML schema and of the meanings of the elements therein defined.

A protocol extension is a set of XML definitions related to the CLUE protocol that is contained in an XML schema which is different from the normative CLUE protocol schema. Such definitions can represent: (i) information to be carried within the existing messages in place of <any> and <anyAttribute> elements; (ii) new messages designed for the CLUE telepresence control. Such XML definitions refer to

information not envisioned during the CLUE protocol design phase. A protocol extension is then represented by a reference to the defining XML schema. If a protocol extension is supported by both a CI and a CR, it means that both are aware of the associated XML schema and of the meanings of the elements defined within it.

When the CR receives the CI's OPTIONS message, it selects the data model extensions and the protocol extensions that it is able to support, and then provides them into the OPTIONS RESPONSE message back to the CI. Only the extensions included in the RESPONSE message can be used during the telepresence session.

The XML schema definition of the OPTIONS message is provided in the following.

```
<!-- CLUE OPTIONS REQUEST -->
<xs:complexType name="optionsMessageType">
<xs:complexContent>
<xs:extension base="clueRequestMessageType">
<xs:sequence>
<!-- optional fields -->
<xs:element ref="options" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE OPTIONS -->
<xs:element name="options" type="optionsType"/>

<xs:complexType name="optionsType">
<xs:sequence>
<xs:element name="dm-exts" type="schemaRefList" minOccurs="0" maxOccurs="1"/>
<xs:element name="protocol-exts" type="schemaRefList" minOccurs="0"
            maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<!-- SCHEMA REF LIST TYPE -->
<xs:complexType name="schemaRefList">
<sequence>
<element name="schemaRef" type="xs:anyURI" maxOccurs="unbounded"/>
</sequence>
</xs:complexType>
```

10.1. An example using OPTIONS

An example of OPTIONS dialogue is provided in the following.

```
+------+              +------+
|  CI  |              |  CR  |
|      |              |      |
+--+---+              +--+---+
   |                     |
   | OPTIONS 3.0         |
   | dm-ext: s1, s2, s3  |
   | pr-ext: s4, s5      |
   |<-------------------+|
   |                     |
   |        RESPONSE 1.0 |
   |        dm-ext: s1   |
   |        pr-ext: -    |
   |+------------------->|
   |                     |
   |                     |
   |             ADV 1.0 |
   |+------------------->|
   |                     |
   |  CONFIGURE+ACK 1.0  |
   |<-------------------+|
   |                     |
   |                     |
   v                     v
```

When the CLUE channel is ready, the CI issues an OPTIONS request to the CR. The CI uses the 3.0 version of the CLUE protocol, and supports schemas s1, s2, s3 as data model extensions and schemas s4, s5 as protocol extensions.

The CR speaks the 1.0 version of the CLUE protocol and supports only the first data model extension among those indicated by the CI. It then issues a v. 1.0 RESPONSE to the CI copying only the supported option. The CI is able to understand that it can use only the 1.0 version of the protocol and the s1 extension.

After the negotiation phase is completed, both CP starts their MP and MC machines and the dialouge for the media sessions set up starts. An example of possible messaging flowing on the channel is represented by the ADV issued by the CI's MP towards the CR's MC and

the following CONF+ACK, both version 1.0.

11. XML Schema of CLUE protocol messages

In this section we paste the XML schema defining the ADVERTISEMENT, CONFIGURE and RESPONSE messages contained in [I-D.kyzivat-clue-signaling]. At the time of writing, it assumes that encodings are described in SDP as m-lines with a text identifier, and that the identifier has the same value as the encodingIDs embedded in the <encodingGroups>. However, that assumption is still under discussion in the context of the CLUE-SDP coupling issues.

*** TO BE UPDATED ***

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
version="0.02"
targetNamespace="urn:ietf:params:xml:ns:clue-message"
xmlns:tns="urn:ietf:params:xml:ns:clue-message"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dm="urn:ietf:params:xml:ns:clue-info"
xmlns="urn:ietf:params:xml:ns:clue-message"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

<!-- Import data model schema -->
<xs:import namespace="urn:ietf:params:xml:ns:clue-info"
schemaLocation="clue-data-model-01.xsd"/>

<!-- ELEMENT DEFINITIONS -->
<xs:element name="response" type="responseMessageType"/>
<xs:element name="advertisement" type="advertisementMessageType"/>
<xs:element name="configure" type="configureMessageType"/>
<xs:element name="readv" type="readvMessageType"/>
<xs:element name="options" type="optionsMessageType"/>

<!-- CLUE MESSAGE TYPE -->
<xs:complexType name="clueMessageType" abstract="true">
<xs:sequence>
<!-- mandatory fields -->
<!-- TBS: version info -->
</xs:sequence>
</xs:complexType>
```

```
<!-- CLUE REQUEST MESSAGE TYPE -->
<xs:complexType name="clueRequestMessageType" abstract="true">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="requestNumber" type="xs:integer"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE OPTIONS REQUEST -->
<xs:complexType name="optionsMessageType">
<xs:complexContent>
<xs:extension base="clueRequestMessageType">
<xs:sequence>
<!-- optional fields -->
<xs:element ref="options" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE OPTIONS -->
<xs:element name="options" type="optionsType"/>

<xs:complexType name="optionsType">
<xs:sequence>
<xs:element name="dm-exts" type="schemaRefList" minOccurs="0" maxOccurs="1"/>
<xs:element name="protocol-exts" type="schemaRefList" minOccurs="0"
            maxOccurs="1"/>
</xs:sequence>
</xs:complexType>

<!-- SCHEMA REF LIST TYPE -->
<xs:complexType name="schemaRefList">
<sequence>
<element name="schemaRef" type="xs:anyURI" maxOccurs="unbounded"/>
</sequence>
</xs:complexType>

<!-- CLUE NOTIFICATION MESSAGE TYPE -->
<xs:complexType name="clueNotificationMessageType" abstract="true">
<xs:complexContent>
<xs:extension base="clueMessageType">
```

```
<xs:sequence>
<!-- mandatory fields -->
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>


<!-- CLUE RESPONSE MESSAGE TYPE -->
<xs:complexType name="clueResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="requestNumber" type="xs:integer"/>
<xs:element name="reason" type="reasonType" minOccurs="1"/>
<!-- optional fields -->
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- RESPONSE MESSAGE TYPE -->
<xs:complexType name="responseMessageType">
<xs:complexContent>
<xs:extension base="clueRequestMessageType">
<xs:sequence>
<!-- mandatory fields -->
<!-- TBD. -->
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>


<!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
<xs:complexType name="advertisementMessageType">
<xs:complexContent>
<xs:extension base="clueNotificationMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="advNumber" type="xs:unsignedInt"/>
<xs:element name="mediaCaptures"
type="dm:mediaCapturesType"/>
<xs:element name="encodingGroups"
type="dm:encodingGroupsType"/>
```

```
<!-- The encodings are defined via identifiers in the SDP,
referenced in encodingGroups -->
<xs:element name="captureScenes"
type="dm:captureScenesType"/>
<!-- optional fields -->
<xs:element name="simultaneousSets"
type="dm:simultaneousSetsType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>


<!-- CLUE CONFIGURE MESSAGE TYPE -->
<xs:complexType name="configureMessageType">
<xs:complexContent>
<xs:extension base="clueRequestMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="advNumber" type="xs:unsignedInt"/>
<!-- optional fields -->
<xs:element name="captureEncodings"
type="dm:captureEncodingsType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- REASON TYPE -->
<xs:complexType name="reasonType">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute type="xs:short" name="code" use="required"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>


</xs:schema>
```

12. Examples

TBD

13. Handling channel errors

TBD

14. Diff with the -02 version

1. "Terminology" section added.

2. Introduced the concept of "CLUE Participant" - an Endpoint or a MCU able to use the CLUE protocol within a telepresence session. A CLUE Participant can act as a Media Provider and/or as a Media Consumer.

3. INtroduced the ACK/NACK mechanism for the ADVERTISEMENT.

4. MP and MC state machines have been updated. The CP state machine has been added.

15. Acknowledgments

We would like to thank Liuyan Scarlett for her precious feedback on the protocol document, as well as for proposing the introduction of the concept of a main state machine including the MP and MC sub state machines.

16. Informative References

[I-D.ietf-clue-data-model-schema] Presta, R. and S. Romano, "An XML Schema for the CLUE data model", draft-ietf-clue-data-model-schema-00 (work in progress), July 2013.

[I-D.ietf-clue-framework] Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", draft-ietf-clue-framework-12 (work in progress), October 2013.

[I-D.ietf-clue-telepresence-requirements] Romanow, A., Botzko, S., and M. Barnes,

"Requirements for Telepresence Multi-Streams", draft-ietf-clue-telepresence-requirements-06 (work in progress), October 2013.

[I-D.kyzivat-clue-signaling] Kyzivat, P., Xiao, L., Groves, C., and R. Hansen, "CLUE Signaling", draft-kyzivat-clue-signaling-05 (work in progress), September 2013.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC4353] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117, January 2008.

[RFC5261] Urpalainen, J., "An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors", RFC 5261, September 2008.

[RFC6502] Camarillo, G., Srinivasan, S., Even, R., and J. Urpalainen, "Conference Event Package Data Format Extension for Centralized Conferencing (XCON)", RFC 6502, March 2012.

[RFC6503] Barnes, M., Boulton, C., Romano, S., and H. Schulzrinne, "Centralized Conferencing Manipulation Protocol", RFC 6503, March 2012.

Authors' Addresses

Roberta Presta
University of Napoli
Via Claudio 21
Napoli 80125
Italy

EMail: roberta.presta@unina.it

Simon Pietro Romano
University of Napoli
Via Claudio 21
Napoli 80125
Italy

EMail: spromano@unina.it