CLUE Working Group R. Presta
Internet-Draft S. Romano
Intended status: Standards Track University of Napoli
Expires: December 21, 2014 June 19, 2014

CLUE protocol
draft-ietf-clue-protocol-00

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 on the SIP/SDP session establishment phase. CLUE messages flow upon the CLUE data channel, based on reliable and ordered SCTP over DTLS transport, as described in [I-D.ietf-clue-datachannel]. Message details, together with the behavior of CLUE Participants acting as Media Providers and/or Media Consumers, are herein discussed.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 21, 2014.

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 fully announce its current telepresence capabilities to a MC in terms of available media captures, groups of encodings, simultaneity constraints and other information envisioned in [I-D.ietf-clue-framework];

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

CLUE-capable endpoints are connected by means of the CLUE data channel, an SCTP over DTLS channel which is opened and established as depicted respectively in [I-D.kyzivat-clue-signaling] and [I-D.kyzivat-clue-signaling]. CLUE protocol messages flowing upon such channel are detailed in the following, both syntactically and semantically.

In Section 3 we provide a general overview of the CLUE protocol. CLUE protocol messages are detailed in Section 4 The CLUE Participant state machine is introduced in Section 5. Versioning and extensions are discussed in Section 7 and Section 8, respectively. The XML schema defining the CLUE messages is reported in Section 9.

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 exploited 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 an endpoint or a 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 a MCU) able to send Media Streams.

Media Consumer A CLUE Participant (i.e., an Endpoint or a MCU) able to receive Media Streams.

3. Overview of the CLUE protocol

The CLUE protocol has been conceived to enable CLUE telepresence session. It is designed in order to address SDP limitations in terms of the description of several information about the multimedia streams that are involved in a real-time multimedia conference. Indeed, by simply using SDP we are not able to convey the information about the features of the flowing multimedia streams that is needed to enable a "being there" rendering. Such information is designed in the CLUE framework document and formally defined and described in the CLUE data model document. The CLUE protocol represents the mechanism that enables the exchange of CLUE information between CLUE Participants. It mainly provides the messages to enable a Media Provider to advertise its telepresence capabilities and to enable a Media Consumer to select the desired telepresence options.

The CLUE protocol, as defined in the following, is a stateful, client-server, XML-based application protocol. CLUE protocol messages flow on realiable and ordered SCTP over DTLS transport channel connecting two CLUE Participants. Messages carries information taken from the XML-based CLUE data model ([I-D.ietf-clue-data-model-schema]). Three main communication layers can be identified:

1. Establishment of the CLUE data channel: in this phase, the CLUE data channel setup takes place. If it ends up successfully, the

CPs are able to communicate and start the initiation phase.

2. Negotiation of the CLUE protocol version and options (initiation phase): the CPs connected via the CLUE data channel agree on the version and on the options to be used during the telepresence session. Special CLUE messages are used for such a task. At the end of that basic negotiation, each CP starts its activity as a CLUE MP and/or CLUE MC.

3. CLUE telepresence capabilities description and negotiation: in this phase, the MP-MC offer-answer dialogues take place on the data channel by means of the CLUE protocol messages.

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. CLUE protocol version numbers are characterized by a major version number and a minor version number, both unsigned integer, separated by a dot. While minor version numbers denote backword compatible changes in the context of a given major version, different major version numbers generally indicate a lack of interoperability between the protocol implementations. In order to correctly establish a CLUE dialogue, the involved CPs MUST have in common a major version number (see Section 7 for further details). The subset of the protocol options and extensions that are allowed within the CLUE session is also determined in the initiation phase, such subset being the one including only the options that are supported by both parties. A mechanism for the negotiation of the CLUE protocol version and extensions is envisioned in the initiation phase. 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) specifying the supported version and extensions. The CR then answers by selecting the subset of the CI extensions that it is able to support and determines the protocol version to be used.

After that negotiation phase is completed, CLUE Participants describe and agree on the media flows to be exchanged. Indeed, being CPs A and B both transmitting and receiving, 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 is 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 its available media captures and encoding capabilities to the MC, as well as its simultaneity constraints, according to the information model defined in [I-D.ietf-clue-framework]. The CLUE message conveing the MP's multimedia offer is the ADVERTISEMENT message. Such message leverages the XML data model definitions provided in [I-D.ietf-clue-data-model-schema].

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

Besides ADVERTISEMENT and CONFIGURE, other messages have been conceived in order to provide all the needed mechanisms and operations and will be detailed in the following sections.

4. Protocol messages

CLUE protocol messages are textual, XML-based messages that enable the configuration of the telepresence session. The formal definition of such messages is provided in the XML Schema provided at the end of this document (Section 9).

The XML definitions of the CLUE information provided in [I-D.ietf-clue-data-model-schema] are included within some CLUE protocol messages (namely the ADVERTISEMENT, the CONFIGURE, and the READV RESPONSE messages), in order to use the concept defined in [I-D.ietf-clue-framework].

The CLUE protocol messages that have been defined up to now are the following:

- OPTIONS
- OPTIONS RESPONSE
- ADVERTISEMENT (ADV)
- ADVERTISEMENT ACKNOWLEDGE (ACK)

- CONFIGURE (CONF)
- CONFIGURE RESPONSE
- READV
- READV RESPONSE

While the OPTIONS and OPTIONS RESPONSE messages are exchanged in the initiation phase between the CPs, the other messages are involved in MP-MC dialogues.

Each CLUE message inherits a basic structure depicted in the following figure:

```
<!-- CLUE MESSAGE TYPE -->
<xs:complexType name="clueMessageType" abstract="true">
<xs:sequence>
<xs:element name="clueId" type="xs:string"/>
<xs:element name="sequenceNr" type="xs:unsignedInt"/>
</xs:sequence>
<xs:attribute name="protocol" type="xs:string" fixed="CLUE" use="required"/>
<xs:attribute name="v" type="xs:string" use="required"/>
</xs:complexType>
```

The basic structure determines the mandatory information that is carried within each CLUE message. Such an information is made by:

- clueId: an XML element containing the identifier of the CP within the telepresence system;
- sequenceNr: an XML element containing the local message sequence number;
- protocol: a mandatory attribute set to "CLUE" identifying the procotol the messages refer to;
- v: a mandatory attribute carrying the version of the protocol

Each CP should manage uo to three streams of sequence numbers: (i) one for the messages exchanged in the initiation phase, (ii) one for the messages exchanged as MP, and (iii) one for the messages exchanged as MC.

4.1. OPTIONS

The OPTIONS message is sent by the CP which is the CI to the CP which is the CR as soon as the CLUE data channel is ready. Besides the information envisioned in the basic structure, it specifies:

- mediaProvider: a mandatory boolean field set to "true" if the CP is able to act as a MP
- mediaConsumer: a mandatory boolean field set to "true" if the CP is able to act as a MC
- supportedVersions: the list of the supported versions
- supportedOptions: the list of the supported options

The XML Schema of such a message is reported below:

```
<!-- CLUE OPTIONS -->
<xs:complexType name="optionsMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="mediaProvider" type="xs:boolean"/>
<xs:element name="mediaConsumer" type="xs:boolean"/>
<xs:element name="supportedVersions" type="versionsListType" minOccurs="0"/>
<xs:element name="supportedOptions" type="optionsListType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- VERSIONS LIST TYPE -->
<xs:complexType name="versionsListType">
<xs:sequence>
<xs:element name="version" type="xs:string" minOccurs="1"
 maxOccurs="unbounded"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- OPTIONS LIST TYPE -->
<xs:complexType name="optionsListType">
<xs:sequence>
<xs:element name="option" type="optionType" minOccurs="1"
 maxOccurs="unbounded"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- OPTION TYPE -->
<xs:complexType name="optionType">
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="schemaRef" type="xs:anyURI" minOccurs="0"/>
<xs:element name="version" type="xs:string" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

<supportedVersions> contains the list of the versions that are supported by the CI. Only one <version> element SHOULD be provided for each major version supported, containing the maximum minor version number of such a version, since all minor versions are backward compatible. If no <supportedVersions> is carried whithin the OPTIONS message, the CI supports only the version declared in the "v" attribute. For example, if the "v" attribute has a value of "3.4" and there is not a <supportedVersions> tag in the OPTIONS message, it means the CI supports only major version 3 with all the minor versions comprised between 3.0 the 3.4 included. If a <supportedVersion> is provided, at least one <version> tag MUST be included.

The <supportedOptions> element specifies the list of the options supported by the CI. If there is no <supportedOptions> in the OPTIONS message, the CI does not support anything more than what is envisioned in the versions it supports. For each option, an <option> element is provided. An option is characterized by a name, an XML schema of reference where the option is defined, and the version of the protocol which the option refers to. [to be discussed: difference between options and extensions]

4.2. OPTIONS RESPONSE

The OPTIONS RESPONSE is sent by a CR to a CI as a reply to the OPTIONS message. As depicted in the figure below, the OPTIONS RESPONSE contains mandatorily a response code and a response string indicating the processing result of the OPTIONS message. Following, the CR attaches two boolean tags, <mediaProvider> and <mediaConsumer>, expressing the supported roles in terms of respectively MP and MC, similarly to what the CI does in the OPTIONS message. Finally, the highest commonly supported version number is expressed in the <version> field and just the commonly supported options in the <commonOptions> field.

```
<!-- CLUE OPTIONS RESPONSE (2 WAY) -->
<xs:complexType name="optionsResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="mediaProvider" type="xs:boolean" minOccurs="0"/>
<xs:element name="mediaConsumer" type="xs:boolean" minOccurs="0"/>
<xs:element name="version" type="xs:string" minOccurs="0"/>
<xs:element name="commonOptions" type="optionsListType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

After the reception of such message, the version to be used is determined by each part of the conversation. Indeed, it is the one provided in the <version> tag of the OPTIONS RESPONSE message. The following CLUE messages will use such a version number in the "v" attribute. The allowed options in the CLUE dialogue will be those indicated in the <commonOptions> of the OPTIONS RESPONSE message.

4.3. ADVERTISEMENT

This message is used by the MP to advertise the available media captures and related information to the MC. The MP sends to the MC an ADV as soon as it is ready after the successful completion of the initiation phase. During the telepresence session, the ADV can be sent from the MP both periodically and on a per-event basis, i.e., each time there are changes in the MP's CLUE telepresence capabilities.

The ADV structure is defined in the picture below. The ADV contains elements compliant with the CLUE data model that characterize the MP's telepresence offer. Namely, such elements are: the list of the media captures (<mediaCaptures>), of the encoding groups (>encodingGroups>), of the capture scenes (>captureScenes>) and of the global capture entries (>globalCaptureEntries>), and the list of the represented participants (>participants>). Each of them is fully described in the CLUE framework document and formally defined in the CLUE data model document.

```
<!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
<xs:complexType name="advertisementMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="mediaCaptures" type="dm:mediaCapturesType"/>
<xs:element name="encodingGroups" type="dm:encodingGroupsType"/>
<xs:element name="captureScenes" type="dm:captureScenesType"/>
<xs:element name="simultaneousSets" type="dm:simultaneousSetsType"
 minOccurs="0"/>
<xs:element name="globalCaptureEntries" type="dm:globalCaptureEntriesType"
 minOccurs="0"/>
<xs:element name="participants" type="dm:participantsType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

[to be discussed: 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]].

4.4. ADVERTISEMENT ACKNOWLEDGEMENT

The ACK message is sent by a MC to a MP to acknowledge an ADV message. As it can be seen from the message schema provided in the following, the ACK contains a response code and a reason string for describing the processing result of the ADV. The <advSequenceNr> carries the sequence number of the ADV the ACK refers to.

```
<!-- ADV ACK MESSAGE TYPE -->
<xs:complexType name="advAcknowledgementMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="advSequenceNr" type="xs:unsignedInt"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

4.5. CONFIGURE

The CLUE CONFIGURE message is sent from a MC to a MP to list the advertised captures the MC wants to receive. The MC can send a CONF after the reception of an ADV or each time it wants to request other captures that have been previously advertised by the MP. The content of the CONF message is shown below.

```
 <!-- CLUE CONFIGURE MESSAGE TYPE -->
<xs:complexType name="configureMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="advSequenceNr" type="xs:unsignedInt"/>
<xs:element name="ack" type="xs:boolean" minOccurs="0" fixed="true"/>
<xs:element name="captureEncodings" type="dm:captureEncodingsType"
 minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

In the >advSequenceNr< element is contained the sequence number of the ADVERTISEMENT or of the READV RESPONSE message the CONFIGURE refers to.

The optional boolean <ack> element, set to "true", if present, indicates that the CONF message also acknowledge the referred advertisement, by applying in that way a piggibacking mechanism for simultaneously acknowledging and replying to the ADV message. The <ack> element SHOULD not be present at all if an ADV ACK message has been already sent back to the MP and if the CONFIGURE refers to a READV RESPONSE message.

The most important content of the CONFIGURE message is the list of the capture encodings provided in the <captureEncodings> element. Such an element is defined in the CLUE data model document and contains a sequence of capture encodings, representing the streams to be instantiated.

4.6. CONFIGURE RESPONSE

```
<!-- CONFIGURE RESPONSE MESSAGE TYPE -->
<xs:complexType name="configureResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="confSequenceNr" type="xs:integer"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

The CONF RESPONSE message is sent from the MP to the MC to communicate the processing result of requests carried in the previously received CONF message. It contains a response code with a reason string indicating either the success or the failure (along with failure details) of a CONF request processing. Following, the <confSequenceNr> field contains the number of the CONF message the response refers to.

4.7. READV

The READV message is a request the MC issues to the MP to retrieve an updated version of the MP's telepresence offer. The content of the READV message is specified in the following.

```
<!-- CLUE READV MESSAGE TYPE -->
<xs:complexType name="readvMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="lastReceivedAdv" type="xs:short"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

The <lastReceivedAdv> element specifies the sequence number of the last ADVERTISEMENT or READV RESPONSE correctly received by the MC.

4.8. READV RESPONSE

The READV RESPONSE is sent by the MP to the MC to reply to a READV message. As shown in the schema below, it contains, besides a response code and a reason string, all the information carried within an ADVERTISEMENT message (media captures, encoding groups, and so on). If there are no updates with respect to the last telepresence offer successfully delivered to the MC (i.e, that having the sequence number specified in the <lastReceiveAdv> field of the READV message), the READV RESPONSE SHOULD carry only the response code with the reason string.

```
<!-- CLUE READV RESPONSE MESSAGE TYPE -->
<xs:complexType name="readvResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="readvSequenceNr" type="xs:string" minOccurs="0"/>
<xs:element name="mediaCaptures" type="dm:mediaCapturesType" minOccurs="0"/>
<xs:element name="encodingGroups" type="dm:encodingGroupsType" minOccurs="0"/>
<xs:element name="captureScenes" type="dm:captureScenesType" minOccurs="0"/>
<xs:element name="simultaneousSets" type="dm:simultaneousSetsType"
minOccurs="0"/>
<xs:element name="globalCaptureEntries" type="dm:globalCaptureEntriesType"
minOccurs="0"/>
<xs:element name="participants" type="dm:participantsType" minOccurs="0"/>
<xs:any namespace="##other"
 processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

4.9. Response codes and reason strings

Examples of response codes and strings are provided in the following table. 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

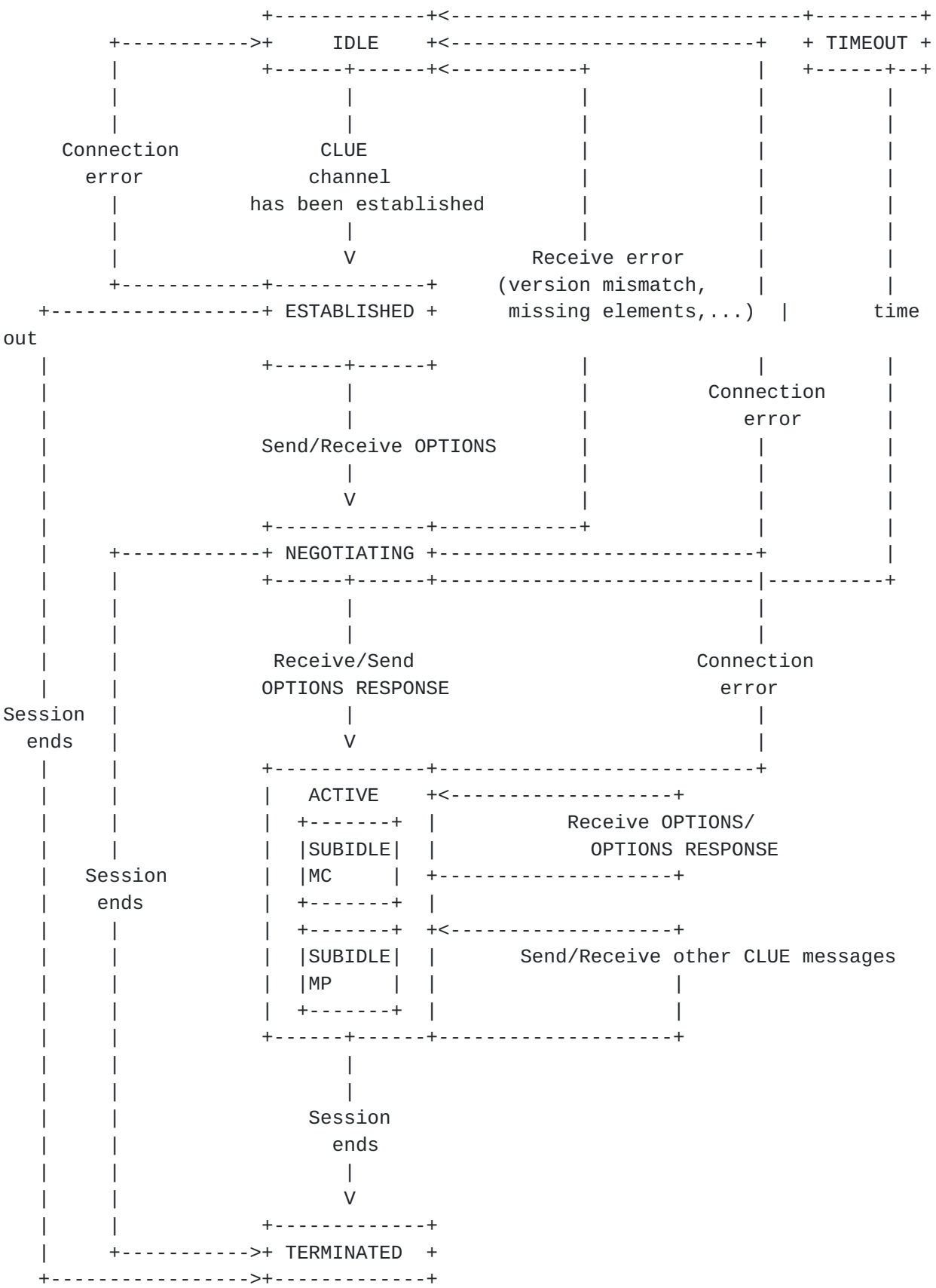
5. Protocol state machines

The CLUE protocol is an application protocol used between two CPs in order to properly configure a multimedia telepresence session. CLUE protocol messages flow upon the CLUE Data Channel, 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 MC process and with the MP process.

6. CLUE Participant's state machine

The main state machines focus on describing the states of CLUE channel from a CLUE channel initiator/receiver. 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 prepares sending an OPTIONS message for version negotiation; otherwise, if the is the Channel Receiver

(CR), it listens to the channel for an OPTIONS message for version negotiation. If an OPTIONS message is sent or is received, the CP moves to the NEGOTIATING state. If the CP checks some error in the request message received, the main state goes back to the IDLE state. [TODO: check this] 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 is received, the main state moves to the ACTIVE state. If the CI checks some error in the OPTIONS RESPONSE message received or receives an OPTIONS RESPONSE indicating an error, it goes back to the IDLE state. When the CP enters in the ACTIVE state, it creates two sub state machines which are the MC state machine and the MP state machine, accordingly to the supported roles. When in the ACTIVE state, if the CP receives a further OPTIONS message for version negotiation or a further OPTIONS RESPONSE messages for version negotiation, it MUST ignore the messages and keep in the ACTIVE state. When in the ACTIVE state, the CP delegates the sending and the processing of the CLUE messages the appropriate MP/MC sub-state machines. 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 due to connection error. [TODO: CLUE messages to cancel/release the session] [TODO: check the diagram]


IDLE
TIMEOUT
Connection error
CLUE channel has been established
Receive error (version mismatch, missing elements,...)
ESTABLISHED
time out
Connection error
Send/Receive OPTIONS
NEGOTIATING
Receive/Send OPTIONS RESPONSE
Connection error
Session ends
ACTIVE
SUBIDLE MC
SUBIDLE MP
Receive OPTIONS/ OPTIONS RESPONSE
Session ends
Send/Receive other CLUE messages
Session ends
TERMINATED

6.1. 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 READV message to the MP ("send RE-ADV"), switching back to the WAIT FOR ADV. Otherwise, the MC moves to the TERMINATED state.

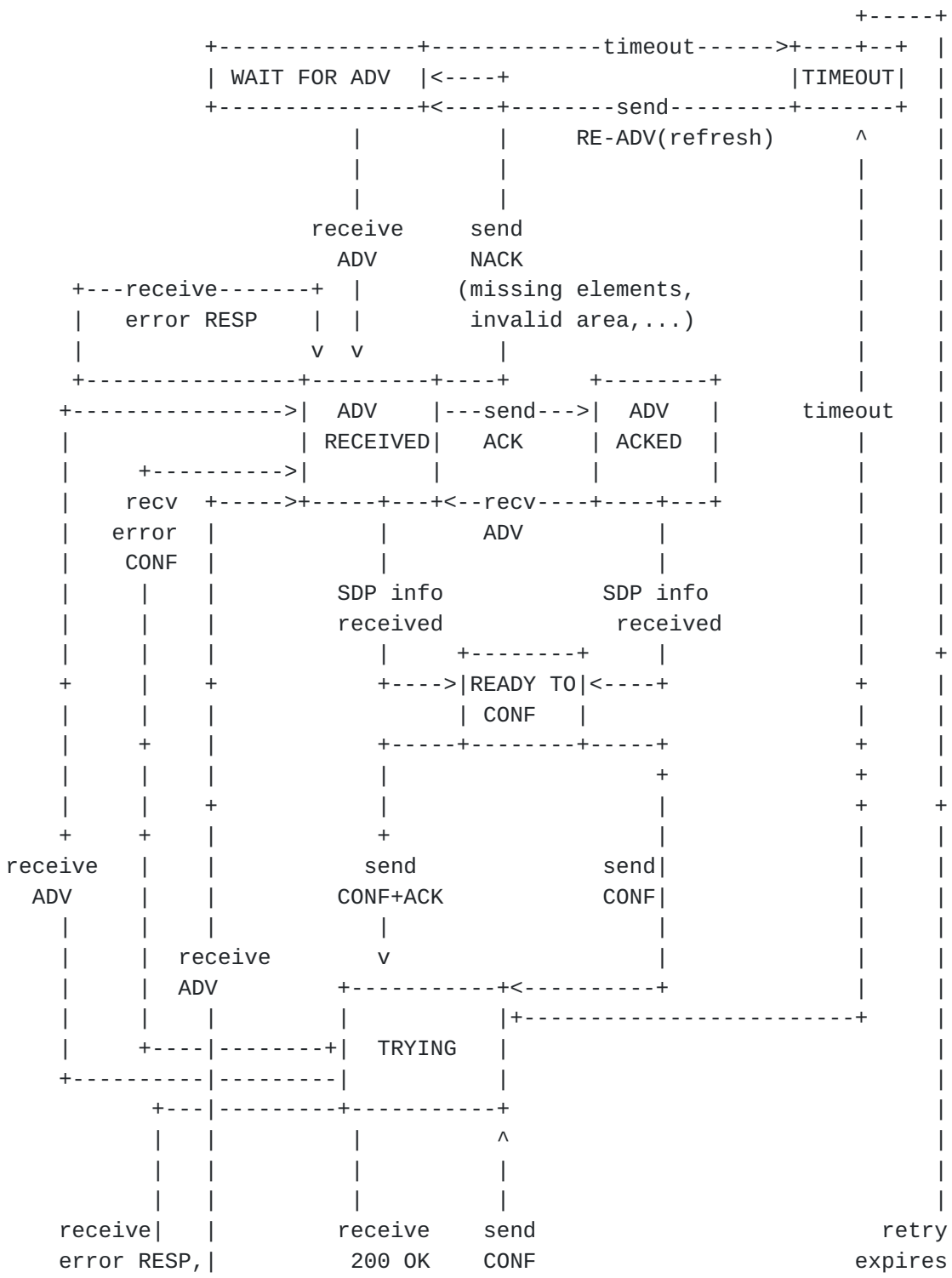
When the ADV has been received ("receive ADV"), the MC goes into the ADV RECEIVED state. The ADV is then parsed. If something goes wrong with the ADV (bad syntax, missing XML elements, etc.), the MC sends a NACK message (an ACK with an error response code) to the MP specifying the encountered problem 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 a successful ACK message to the MP and moves to the ADV ACKED state. When the SDP information arrives, from the ADV RECEIVED or the ADV ACKED state the MC switches to the READY TO CONF state. When the CONF request is ready, the MC sends it and moves to the TRYING state. If the ADV has not been already sent, the MC can piggyback the ACK message within the CONF request.

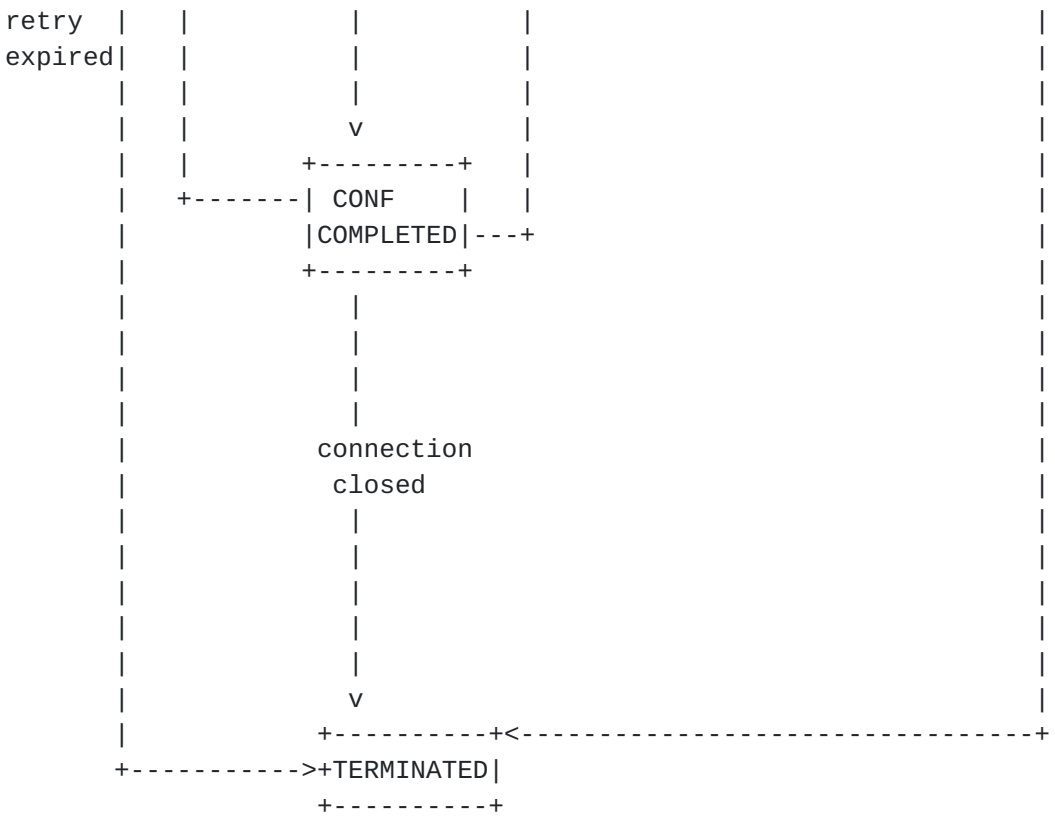
While in the TRYING state, the MC is waiting for a CONF RESPONSE message (to the issued CONF) from the MP. If the timeout expires ("timeout"), the MC moves to the TIMEOUT state and sends a READV in order to solicit a new ADV from the MP. If a CONF RESPONSE with an error code is received ("receive 4xx, 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 CONF 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 RECEIVED 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.





6.2. 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.

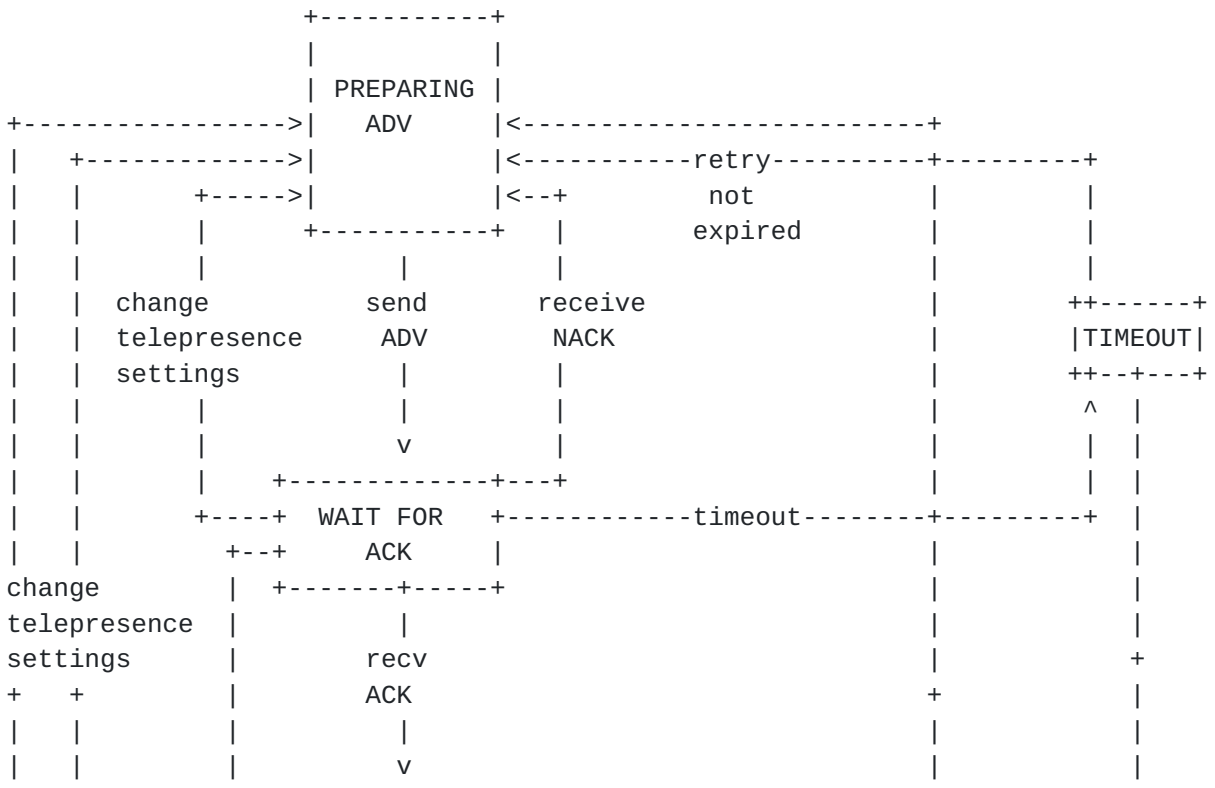
When in the WAIT FOR CONF state, the MP is listening to the channel for a CONF coming from the MC. If a RE-ADV is received, the MP goes back to the IDLE state and issues an ADV again. If telepresence settings change in the meanwhile, it moves back to the PREPARING ADV state and prepares a new ADV to 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.

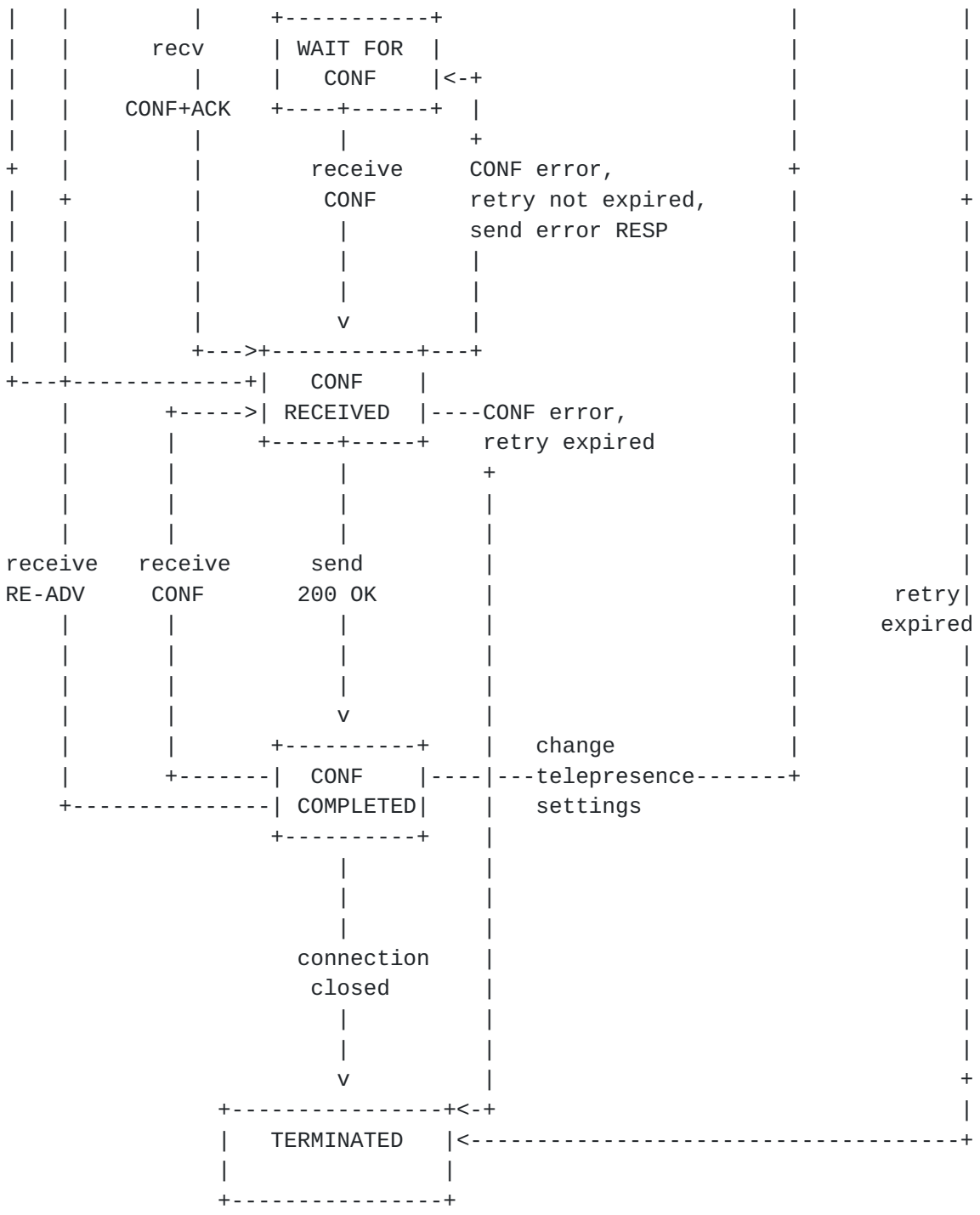
In the TIMEOUT state, if the number of trials does not exceed the retry threshold, the MC comes back to the PREPARING ADV state for sending a new ADV. Otherwise, it goes to the TERMINATED state.

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.





7. 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 version usage is similar in philosophy to XMPP (RFC6120). A version number has major and minor components, each a non-negative integer. Major version changes denote non-interoperable changes. Minor version changes denote schema changes that are backward compatible by ignoring unknown XML elements, or other backward compatible changes.

The minor versions of the XML schema MUST 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 1.5 of the protocol while the MC only understands version 1.4, the MP should have no problem in reverting the dialogue to version 1.4 without exploiting 1.5 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.

8. Extensions and options

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 and extensions can be designed. Extensions specify information and behaviors that are not described in a certain version of the protocol. They can relate to:

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.

As to the first type of extensions, it is possible to distinguish between protocol specific- and data model information. Indeed, CLUE messages are envelopes carrying both:

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

(ii) 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.

On the other hand (second type of extensions), "extra" CLUE protocol messages, i.e., messages not envisioned in the last standard version of the schema, can be needed. In that case, the messages and the associated behavior should be defined in external documents that both the communication parties must be aware of.

Both the types of extensions, i.e., the information and the protocol

extensions, can be characterized by:

a name;

an external XML Schema defining the XML information and/or the XML messages representing the extension;

the standard version of the protocol the extension refers to.

For that reason, the extensions can be represented by means of the <option> element as defined below, which is carried within the OPTIONS and OPTIONS RESPONSE messages to represent the extensions supported by the CI and by the CR.

```
<!-- OPTION TYPE -->
<xs:complexType name="optionType">
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="schemaRef" type="xs:anyURI" minOccurs="0"/>
<xs:element name="version" type="xs:string" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

9. XML Schema

In this section, the XML schema defining the CLUE messages is provided.

```
<?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="data-model-schema-05.xsd"/>

<!-- ELEMENT DEFINITIONS -->
<xs:element name="options" type="optionsMessageType"/>
<xs:element name="optionsResponse" type="optionsResponseMessageType"/>
<!--<xs:element name="optionsAck" type="optionsAcknowledgementMessageType"/>-->
<xs:element name="advertisement" type="advertisementMessageType"/>
<xs:element name="ack" type="advAcknowledgementMessageType"/>
<xs:element name="configure" type="configureMessageType"/>
<xs:element name="configureResponse" type="configureResponseMessageType"/>
<xs:element name="readv" type="readvMessageType"/>
<xs:element name="readvResponse" type="readvResponseMessageType"/>

<!-- CLUE MESSAGE TYPE -->
<xs:complexType name="clueMessageType" abstract="true">
<xs:sequence>
<xs:element name="clueId" type="xs:string"/>
<xs:element name="sequenceNr" type="xs:unsignedInt"/>
</xs:sequence>
<xs:attribute name="protocol" type="xs:string" fixed="CLUE" use="required"/>
<xs:attribute name="v" type="xs:string" use="required"/>
</xs:complexType>

<!-- CLUE OPTIONS -->
<xs:complexType name="optionsMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="mediaProvider" type="xs:boolean"/>
<xs:element name="mediaConsumer" type="xs:boolean"/>
<xs:element name="supportedVersions" type="versionsListType" minOccurs="0"/>
<xs:element name="supportedOptions" type="optionsListType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- VERSIONS LIST TYPE -->
<xs:complexType name="versionsListType">
<xs:sequence>
<xs:element name="version" type="xs:string" minOccurs="1"
 maxOccurs="unbounded"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>
```

```
<!-- OPTIONS LIST TYPE -->
<xs:complexType name="optionsListType">
<xs:sequence>
<xs:element name="option" type="optionType" minOccurs="1"
 maxOccurs="unbounded"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- OPTION TYPE -->
<xs:complexType name="optionType">
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="schemaRef" type="xs:anyURI" minOccurs="0"/>
<xs:element name="version" type="xs:string" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:complexType>

<!-- CLUE OPTIONS RESPONSE (2 WAY) -->
<xs:complexType name="optionsResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="mediaProvider" type="xs:boolean" minOccurs="0"/>
<xs:element name="mediaConsumer" type="xs:boolean" minOccurs="0"/>
<xs:element name="version" type="xs:string" minOccurs="0"/>
<xs:element name="commonOptions" type="optionsListType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE OPTIONS RESPONSE (3 WAYS) -->
<!-- <xs:complexType name="optionsResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="mediaProvider" type="xs:boolean"/>
<xs:element name="mediaConsumer" type="xs:boolean"/>
<xs:element name="supportedVersions" type="versionsListType"
```

```
 minOccurs="0"/>
<xs:element name="supportedOptions" type="optionsListType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
 -->

<!-- CLUE OPTIONS ACK (3 WAYS)-->
<!--
<xs:complexType name="optionsAckMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="version" type="xs:string" minOccurs="0"
 maxOccurs="1"/>
<xs:element name="commonOptions" type="supportedOptionsType" minOccurs="0"/>
<xs:any namespace="##other"
processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
 -->

<!-- CLUE ADVERTISEMENT MESSAGE TYPE -->
<xs:complexType name="advertisementMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="mediaCaptures" type="dm:mediaCapturesType"/>
<xs:element name="encodingGroups" type="dm:encodingGroupsType"/>
<xs:element name="captureScenes" type="dm:captureScenesType"/>
<xs:element name="simultaneousSets" type="dm:simultaneousSetsType"
 minOccurs="0"/>
<xs:element name="globalCaptureEntries" type="dm:globalCaptureEntriesType"
 minOccurs="0"/>
<xs:element name="participants" type="dm:participantsType" minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
```

```
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- ADV ACK MESSAGE TYPE -->
<xs:complexType name="advAcknowledgementMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="advSequenceNr" type="xs:unsignedInt"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE CONFIGURE MESSAGE TYPE -->
<xs:complexType name="configureMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<!-- mandatory fields -->
<xs:element name="advSequenceNr" type="xs:unsignedInt"/>
<xs:element name="ack" type="xs:boolean" minOccurs="0" fixed="true"/>
<xs:element name="captureEncodings" type="dm:captureEncodingsType"
 minOccurs="0"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CONFIGURE RESPONSE MESSAGE TYPE -->
<xs:complexType name="configureResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="confSequenceNr" type="xs:integer"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
```

```
</xs:complexContent>
</xs:complexType>

<!-- CLUE READV MESSAGE TYPE -->
<xs:complexType name="readvMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="lastReceivedAdv" type="xs:short"/>
<xs:any namespace="##other" processContents="lax" minOccurs="0"/>
</xs:sequence>
<xs:anyAttribute namespace="##other" processContents="lax"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>

<!-- CLUE READV RESPONSE MESSAGE TYPE -->
<xs:complexType name="readvResponseMessageType">
<xs:complexContent>
<xs:extension base="clueMessageType">
<xs:sequence>
<xs:element name="responseCode" type="xs:short"/>
<xs:element name="reasonString" type="xs:string"/>
<xs:element name="readvSequenceNr" type="xs:string" minOccurs="0"/>
<xs:element name="mediaCaptures" type="dm:mediaCapturesType" minOccurs="0"/>
<xs:element name="encodingGroups" type="dm:encodingGroupsType" minOccurs="0"/>
<xs:element name="captureScenes" type="dm:captureScenesType" minOccurs="0"/>
<xs:element name="simultaneousSets" type="dm:simultaneousSetsType"
minOccurs="0"/>
<xs:element name="globalCaptureEntries" type="dm:globalCaptureEntriesType"
minOccurs="0"/>
<xs:element name="participants" type="dm:participantsType" minOccurs="0"/>
<xs:any namespace="##other"
 processContents="lax" minOccurs="0"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>

</xs:schema>
```

10. Diff with draft-presta-clue-protocol-04

1. The response code type error in the OPTIONS response (and in other parts) has been corrected.

11. Diff with draft-presta-clue-protocol-03

1. The XML Schema has been deeply revised and completed.

2. The descriptions of the CLUE messages have been added.

3. The distinction between major version numbers and minor version numbers has been cut and pasted from [I-D.kyzivat-clue-signaling].

4. Besides the two way one, a three way mechanism for the options negotiation has been proposed and provided to foster discussion.

12. Diff with draft-presta-clue-protocol-02

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.

13. Acknowledgments

The authors thank all the CLUErs for their precious feedbacks and support, in particular Paul Kyzivat, Christian Groves and Scarlett Liuyan.

14. 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-04 (work in progress), March 2014.

[I-D.ietf-clue-datachannel] Holmberg, C., "CLUE Protocol Data Channel", draft-ietf-clue-datachannel-00 (work in progress), March 2014.

[I-D.ietf-clue-framework] Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", draft-ietf-clue-framework-15 (work in progress), May 2014.

[I-D.ietf-clue-telepresence-requirements] Romanow, A., Botzko, S., and M. Barnes, "Requirements for Telepresence Multi-Streams", draft-ietf-clue-telepresence-requirements-07 (work in progress), December 2013.

[I-D.kyzivat-clue-signaling] Kyzivat, P., Xiao, L., Groves, C., and R. Hansen, "CLUE Signaling", draft-kyzivat-clue-signaling-08 (work in progress), April 2014.

[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