CCMP: a novel standard protocol for Conference Management in the XCON Framework

Mary Barnes
Nortel
mary.barnes@nortel.com

Lorenzo Miniero
Meetecho srl
lorenzo@meetecho.com

Roberta Presta
University of Napoli Federico II
roberta.presta@unina.it

Simon Pietro Romano
Univeristy of Napoli Federico II
spromano@unina.it

Henning Schulzrinne
Columbia University
hgs+xcon@cs.columbia.edu

ABSTRACT

This paper presents the design and implementation of CCMP, a conference management protocol currently under standardization within the IETF, conceived at the outset as a lightweight protocol allowing conferencing clients to access and manipulate objects describing a centralized conference. The CCMP is a state-less, XML-based, client-server protocol carrying in its request and response messages conference information in the form of XML documents and fragments conforming to the centralized conferencing data model schema. It represents a powerful means to control basic and advanced conference features such as conference state and capabilities, participants and relative roles and details. We first focus on the design of the protocol and then discuss how it has been integrated in the Meetecho collaborative framework developed at the University of Napoli as an active playground for IETF standardization activities in the field of real-time applications and infrastructure.

Categories and Subject Descriptors

H.4.3 [**Information Systems Applications**]: Communications Applications—*Computer conferencing, teleconferencing, and videoconferencing*; C.2.2 [**Computer Communication Networks**]: Network Protocols—*Applications*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Client/server*

General Terms

Standardization, Design, Experimentation

Keywords

Conferencing, Conference Control and Manipulation, Protocol Design, Protocol Integration

1. INTRODUCTION

In the latest years, the IETF *(Internet Engineering Task Force)* has devoted many efforts to the definition of standard conferencing solutions. Among such solutions, the Framework for Centralized Conferencing [2] (XCON Framework) defines a signaling-agnostic architecture, naming conventions and logical entities required for building advanced conferencing systems. The XCON Framework introduces the *conference object* as a logical representation of a conference instance, representing the current state and capabilities of a conference. The Centralized Conferencing Manipulation Protocol (CCMP) illustrated in this paper is the latest output to be produced by the XCON working group. It is currently undergoing review from the international research community and it is heading towards completion and publication as an RFC (Request For Comments) standard document.

CCMP allows authenticated and authorized users to create, manipulate and delete conference objects. Operations on conferences include adding and removing participants, changing their roles, as well as adding and removing media streams and associated end points. CCMP is based on a client-server paradigm and is specifically suited to serve as a conference manipulation protocol within the XCON framework, with the Conference Control Client and Conference Control Server acting as client and server, respectively. The CCMP uses HTTP as the protocol to transfer requests and responses, which contain the domain-specific XML-encoded data objects defined in [7].

This paper is structured in 8 sections. We first briefly introduce, in section 2, the general architecture for centralized conferencing defined by the XCON working group in the IETF. We then present, in section 3, a bird's eye view of the Centralized Conferencing Manipulation Protocol. The same section also provides some insights on the history of the overall specification process. Section 4 drills down on the specific messages that can be carried inside the body of the CCMP protocol, while section 5 concludes the part associated with our standardization work by depicting a typical call flow related to a CCMP-based interaction between a conferencing client and an XCON Conferencing server. The second part of the paper is entirely devoted to the implementation of the CCMP specification. Such part is based on the work ongoing at the University of Napoli "Federico II", which is since long involved in the IETF activities falling in the area of real-time applications and infrastructures. The University of Napoli has contributed to the activities in the

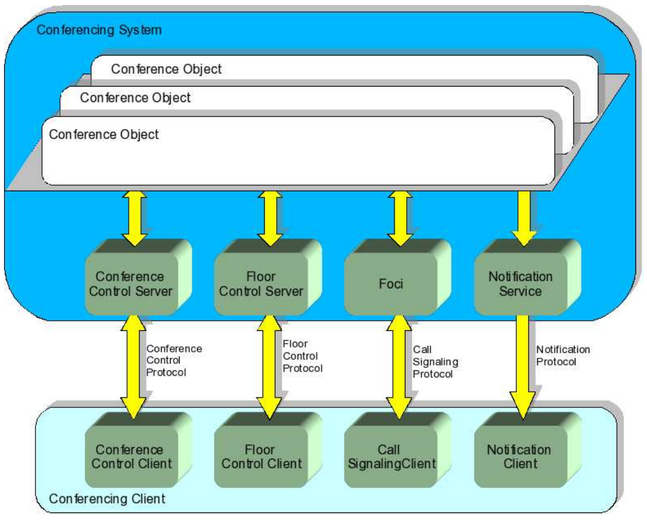


Figure 1: The XCON framework: protocols

XCON working group, by also providing timely prototype implementations of most of the protocols therein involved and/or specified. As far as the CCMP protocol is concerned, we have worked both on the specification of the protocol and on its implementation, during the various phases of its long-lived design history. Information about this activity is hence provided in section 6. Section 7 reports information about the history of the CCMP specification within the IETF community. Finally, section 8 provides some concluding remarks, as well as information about our future work related to the protocol.

2. XCON CONFERENCE CONTROL SYSTEM ARCHITECTURE

RFC5239 defines an architecture for centralized conferencing, and the associated protocol interactons. Such relations are depicted in Fig. 1.

As it can be seen in the figure, several protocols are involved in an XCON-compliant framework architecture. While all the protocols implicitly interact with conference objects somehow, the generically called Conference Control Protocol is probably the most important of them in that regard, as it directly manipulates the conference objects themselves.

Fig. 2 illustrates the typical life cycle of a conference object in the XCON framework. At each instant in time, a conference object is associated with an XML representation compliant with the XCON data model specification. Without digging into the details of the data model, we nevertheless recall that it basically describes all of the features of a conference, starting from its general description (purpose, hosting entity, status, etc.) and arriving at much more detailed information like participants and available media, as well as potential *sidebars* associated with it (i.e. subconferences involving part of the users participating in the main conference).

Creation of such an object is usually performed through a cloning operation, i.e. by replicating the structure of one of the blueprints (also known as conference object templates) available at the server.

A newly created conference object is typically marked as "registered" until the first user joins the conference and it will stay "active" until either the last user leaves the conference (in which case it comes back to the "registered" state) or a user (holding the right to do so) deletes it.

CCMP is the protocol used to manipulate conference objects during the above described lifetime. The next section will present a protocol overview in more detail.

3. PROTOCOL OVERVIEW

CCMP is a client-server, XML-based protocol, which has been specifically conceived to provide users with the necessary means for the creation, retrieval, modification and deletion of conference objects. CCMP is also state-less, which means implementations can safely handle transactions independently from each other. Conference-related information is encapsulated into CCMP messages in the form of XML documents or XML document fragments compliant with the XCON data model representation.

The core set of objects manipulated in the CCMP protocol includes conference blueprints, conference objects, users, and sidebars. CCMP is completely independent from underlying protocols, which means that there can be different ways to carry CCMP messages across the network, from a conferencing client to a conferencing server. Indeed, there have been a number of different proposals as to the most suitable transport solution for the CCMP. It was soon recognized that operations on conference objects can be implemented in many different ways, including remote procedure calls based on SOAP [6] and by defining resources following a RESTful [5] architecture. In both approaches, servers will have to recreate their internal state representation of the object with each update request, checking parameters and triggering function invocations. In the SOAP approach, it would be possible to describe a separate operation for each atomic element, but that would greatly increase the complexity of the protocol. A coarser-grained approach to the CCMP does require that the server process XML elements in updates that have not changed and that there can be multiple changes in one update. For CCMP, the resource (REST) model might appear more attractive, since the conference operations nicely fit the so-called *CRUD (Create-Retrieve-Update-Delete)* approach. Neither of these approaches was finally selected. SOAP was not considered to be general purpose enough for use in a broad range of operational environments. Similarly, it was deemed quite awkward to apply a RESTful approach since CCMP requires a more complex request/response protocol in order to maintain the data both in the server and at the client. This doesn't map very elegantly to the basic request/response model, whereby a response typically indicates whether the request was successful or not, rather than providing additional data to maintain the synchronization between the client and server views. Apart from this, the RESTful approach was considered too restrictive, since it strictly couples the application-level protocol to HTTP messages and semantics. Even though the current implementation of the CCMP relies on HTTP as the preferred transport means, its specification has been kept completely independent of such a choice. Just as an example, work is in full swing at our laboratory related to both an XMPP-based and a UDP-based implementation of the protocol.

The solution for the CCMP at which we arrived can be viewed as a good compromise amongst the above mentioned

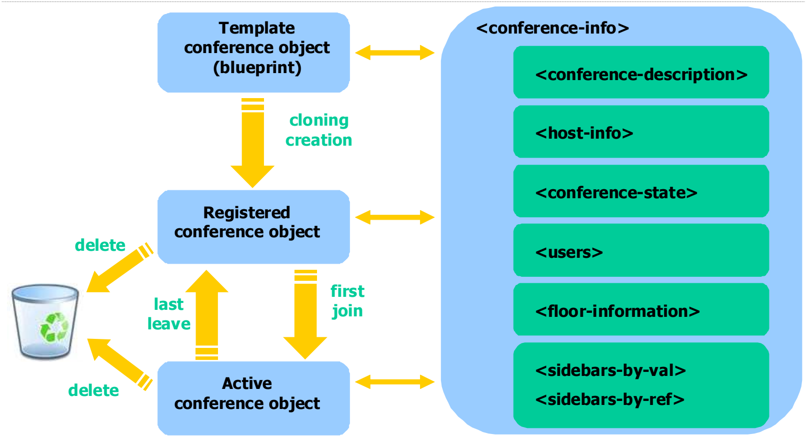


Figure 2: Conference Object Life Cycle

candidates and is referred to as "HTTP single-verb transport plus CCMP body". With this approach, CCMP is able to take advantage of existing HTTP functionality. As with SOAP, it uses a "single HTTP verb" for transport (i.e. a single transaction type for each request/response pair); this allows decoupling CCMP messages from HTTP messages. Similarly, as with any RESTful approach, CCMP messages are inserted directly in the body of HTTP messages, thus avoiding any unnecessary processing and communication burden associated with further intermediaries. This said, we nonetheless remark once again that with this approach no modification to the CCMP messages/operations is required to use a different transport protocol. The remainder of this paper focuses on the selected approach. We will show how the CCMP protocol inserts XML-based CCMP requests into the body of HTTP POST operations and retrieves responses from the body of HTTP "200 OK" messages. CCMP messages will have a MIME-type of "application/ccmp+xml", which appears inside both the "Content-Type" and "Accept" fields of HTTP requests and responses.

3.1 Protocol Operations

The main operations provided by CCMP belong in four general categories:

- create: for the creation of a conference, a conference user, a sidebar, or a blueprint;
- retrieve: to get information about the current state of either a conference object (be it an actual conference or a blueprint, or a sidebar) or a conference user. A retrieve operation can also be used to obtain the XCON-URIs of the current conferences (active or registered) handled by the conferencing server and/or the available blueprints;
- update: to modify the current features of a specified conference or conference user;
- delete: to remove from the system a conference object or a conference user.

Thus, the main targets of CCMP operations are: (i) conference objects associated with either active or registered conferences; (ii) conference objects associated with blueprints; (iii) conference objects associated with sidebars, both embedded in the main conference (i.e. `<entry>` elements in `<sidebars-by-value>`) and external to it (i.e. whose XCON-URIs are included in the `<entry>` elements of <sidebars-by-ref>); (iv) `<user>` elements associated with conference users; (v) the list of XCON-URIs related to conferences and blueprints available at the server, for which only retrieval operations are allowed.

Each operation in the protocol model is atomic and either succeeds or fails as a whole. The conference server must ensure that the operations are atomic in that the operation invoked by a specific conference client completes prior to another client's operation on the same conference object. The details for this data locking functionality are out of scope for the CCMP protocol specification and are implementation specific for a conference server. Thus, the conference server first checks all the parameters, before making any changes to the internal representation of the conference object.

Also, since multiple clients can modify the same conference objects, conference clients should first obtain the current object from the conference server and then update the relevant data elements in the conference object prior to invoking a specific operation on the conference server. In order to effectively manage modifications to conference data, a versioning approach is exploited in the CCMP. More precisely, each conference object is associated with a version number indicating the most up to date view of the conference at the server's side. Such version number is reported to the clients when answering their requests. A client willing to make modifications to a conference object has to send an update message to the server. In case the modifications are all successfully applied, the server sends back to the client a "success" response which also carries information about the current server-side version of the modified object. With such approach, a client which is working on version "X" of a conference object and finds inside a "success" response a version number which is "X+1" can be sure that the version it was aware of was the most up to date. On the other hand, if the "success" response carries back a version which is at least "X+2", the client can detect that the object that has been modified at the server's side was more up to date than the one it was working upon. This is clearly due to the effect of concurrent modification requests issued by independent clients. Hence, for the sake of having available the latest version of the modified object, the client can send to the conference server a further "retrieve" request. In no case a copy of the conference object available at the server is returned to the client as part of the update response message. Such a copy can always be obtained through an ad-hoc "retrieve" message. Based on the above considerations, all CCMP response messages carrying in their body a conference document (or a fragment of it) must contain a "version" parameter. This does not hold for request messages, for which the "version" parameter is not at all required, since it represents useless information for the server: as long as the required modifications can be applied to the target conference object with no conflicts, the server does not care whether or not the client had an up to date view of the information stored at its side. This said, it stands clear that a client which has subscribed at the server, through the XCON event package [4], to notifications about conference object modifications, will always have the most up to date version of that object available at his side.

A final consideration concerns the relation between the CCMP and the main entities it manages, i.e. conference objects. Such objects have to be compliant with the XCON data-model, which identifies some elements and attributes as mandatory. From the CCMP standpoint this can become a problem in cases of client-initiated operations, like either the creation or the update of conference objects. In such cases, not all of the mandatory data can be known in advance to the client issuing a CCMP request. As an example, a client has no means to know, at the time it issues a conference creation request, the XCON-URI that the server will assign to the yet-to-be-created conference and hence it is not able to appropriately fill with that value the mandatory 'entity' attribute of the conference document contained in the request. To solve this kind of issues, the CCMP will fill all mandatory data model fields, for which no value is available at the client at the time the request is constructed, with fake values in the form of wildcard strings (e.g. AUTO_GENERATE_X, with X being an incremental index initialized to a value of 1). Upon reception of the mentioned kinds of requests, the server will: (i) generate the proper identifier(s); (ii) produce a response in which the received fake identifier(s) carried in the request has (have) been replaced by the newly created one(s). With this approach we maintain compatibility with the data model requirements, at the same time allowing for client-initiated manipulation of conference objects at the server's side (which is, by the way, one of the main goals for which the CCMP protocol has been conceived at the outset).

4. CCMP MESSAGES

As anticipated, CCMP is a request/response protocol. Besides, it is completely stateless, which explains why HTTP has been chosen as the perfect transport candidate for it.

For what concerns the protocol by itself, both requests and responses are formatted basically in the same way, as depicted in Fig. 3. In fact, they both have a series of heading parameters, followed by a specialized message indicating the particular request/response (e.g., a request for a specific blueprint). This makes it quite easy to handle a transaction in the proper way and map requests and related responses accordingly.

For what concerns the shared parameters:

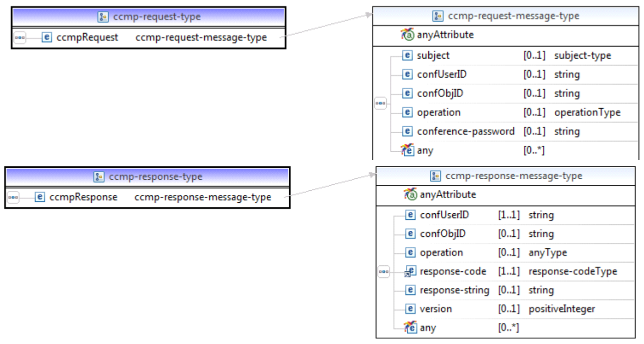

ccmp-request-type
ccmpRequest ccmp-request-message-type
ccmp-request-message-type
anyAttribute
subject [0..1] subject-type
confUserID [0..1] string
confObjID [0..1] string
operation [0..1] operationType
conference-password [0..1] string
any [0..*]
ccmp-response-type
ccmpResponse ccmp-response-message-type
ccmp-response-message-type
anyAttribute
confUserID [1..1] string
confObjID [0..1] string
operation [0..1] anyType
response-code [1..1] response-codeType
response-string [0..1] string
version [0..1] positiveInteger
any [0..*]

Figure 3: CCMP Request and Response messages

- `confUserID` indicates the participant making the request;
- `confObjID` indicates the conference the request is associated with;
- operation specifies what has to be done, according to the specialized message that follows.

Other parameters are defined which are more strictly related to either requests or responses. There is, for instance, a 'password' parameter participants may need to provide in CCMP requests for password-protected conferences, as well as a 'response-code' parameter (which is carried just by responses) providing information about the result of a requested operation.

That said, the core of a CCMP message is actually the specialized part. In fact, as stated in the previous section, the CCMP specification describes several different operations that can be made on a conference object, namely: (i) blueprints retrieval, (ii) conference creation and manipulation, (iii) users management, (iv) sidebar-related operations. All these operations have one or more specialized message formats, instead of a generalized syntax, in order to best suit the specific needs each operation may have.

Indeed, requesting a blueprint and adding a new user to a conference have very different requirements for what concerns the associated semantics level, and as such they need different modes of operation. This is reflected in what is carried in the specialized message body, which will always contain information (compliant with the XCON common data model specification) strictly related to the operation it is associated with. The specialization of the message then allows for an easier and faster management at the implementation level.

To better highlight the considerations above, we show in Fig. 4 the structure of a CCMP `confRequest` message, which is used in all operations concerning the manipulation and control of an entire conference object. As described in the picture, each such message is a specialization of the general CCMP request message, specifically conceived to transport, through the `confInfo` element, an XCON-compliant conference object (i. e. an object whose representation conforms to the common data model specification) towards the CCMP server.

5. CCMP SAMPLE CALL FLOW

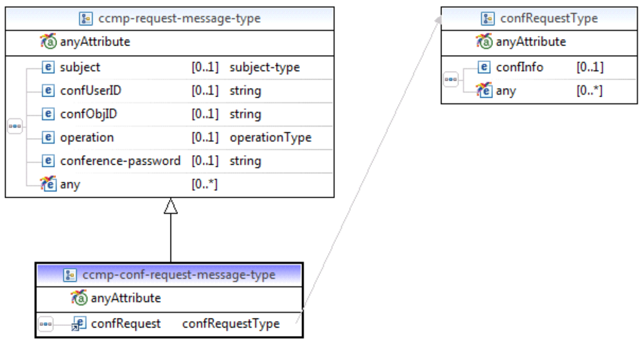

ccmp-request-message-type
anyAttribute
subject [0..1] subject-type
confUserID [0..1] string
confObjID [0..1] string
operation [0..1] operationType
conference-password [0..1] string
any [0..*]
confRequestType
anyAttribute
confInfo [0..1]
any [0..*]
ccmp-conf-request-message-type
anyAttribute
confRequest confRequestType

Figure 4: CCMP confRequest message

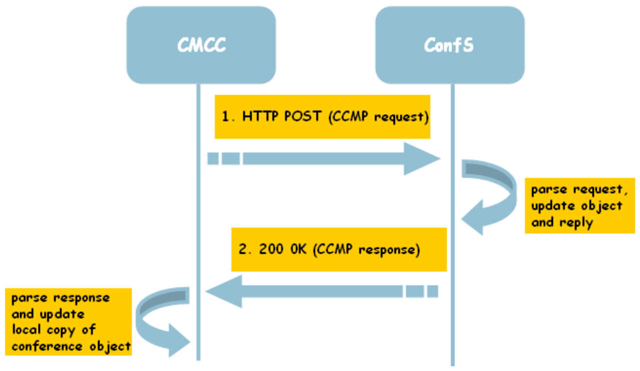

CMCC
ConfS
1. HTTP POST (CCMP request)
parse request, update object and reply
2. 200 OK (CCMP response)
parse response and update local copy of conference object

Figure 5: CCMP transported in HTTP

To better clarify how a CCMP transaction can occur, this section presents a sample call flow. This example comes from a real implementation deployment, as it will be explained in section 6.

For the sake of conciseness, we chose a very simple example, which nevertheless provides the reader with a general overview of both CCMP requests and responses. As mentioned previously, HTTP is suggested by the CCMP specification as a transport for the protocol messages, and Fig. 5 shows the typical request/response paradigm involved in that case.

As it can be seen, the CCMP request (in this case, a 'blueprintRequest') is sent by an interested participant to the conference server. This request is carried as payload of an HTTP POST message:

```
POST /Xcon/Ccmp HTTP/1.1
Content-Length: 657
Content-Type: application/ccmp+xml
Host: example.com:8080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.0.1 (java 1.5)

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
  <ccmpRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:type="ccmp:ccmp-blueprint-request-message-type">
        <confUserID>xcon-userid:Alice@meetecho.com</confUserID>
        <confObjID>xcon:MeetechoRoom@meetecho.com</confObjID>
        <operation>retrieve</operation>
        <ccmp:blueprintRequest/>
  </ccmpRequest>
</ccmp:ccmpRequest>
```

The Content-Type header instructs the receiver that the content of the message is a CCMP message (application/ccmp+xml). For what concerns the request itself, as mentioned, it is a 'blueprintRequest': this means that the participant is interested in the details of a specific blueprint available at the server. This is reflected by the specialized part of the message, i.e., the <ccmp:blueprintRequest> element. The generic parameters introduced in the previous section are also provided as part of the request: 'confUserID' refers to the requestor (Alice's XCON URI), 'confObjID' in this case relates to the blueprint to be retrieved (as an XCON conference URI), while 'operation' clarifies what needs to be done according to the request (retrieve the blueprint).

The CCMP response, in turn, is carried as payload of an HTTP 200 OK reply to the previous POST:

```
HTTP/1.1 200 OK
X-Powered-By: Servlet/2.5
Server: Sun GlassFish Communications Server 1.5
Content-Type: application/ccmp+xml;charset=ISO-8859-1
Content-Length: 1652
Date: Thu, 04 Feb 2010 14:47:56 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ccmp:ccmpResponse
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
  <ccmpResponse xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:type="ccmp:ccmp-blueprint-response-message-type">
    <confUserID>xcon-userid:Alice@meetecho.com</confUserID>
    <confObjID>xcon:MeetechoRoom@meetecho.com</confObjID>
    <operation>retrieve</operation>
    <response-code>200</response-code>
    <response-string>Success</response-string>
    <ccmp:blueprintResponse>
      <blueprintInfo entity="xcon:MeetechoRoom@meetecho.com">
        <info:conference-description>
           <info:display-text>MeetechoRoom</info:display-text>
           <info:available-media>
             <info:entry label="audioLabel">
                 <info:type>audio</info:type>
             </info:entry>
             <info:entry label="videoLabel">
                 <info:type>video</info:type>
             </info:entry>
             <info:entry label="jSummitLabel">
                 <info:type>whiteboard</info:type>
             </info:entry>
             </info:available-media>
        </info:conference-description>
        <info:users>
           <xcon:join-handling>
                 allow
           </xcon:join-handling>
        </info:users>
        <xcon:floor-information>
          <xcon:floor-request-handling>
                confirm
          </xcon:floor-request-handling>
          <xcon:conference-floor-policy>
                <xcon:floor id="audioFloor">
                  <xcon:media-label>
                        audioLabel
                  </xcon:mediaLabel>
                </xcon:floor>
                <xcon:floor id="videoFloor">
                  <xcon:media-label>
                        videoLabel
                  </xcon:mediaLabel>
                </xcon:floor>
                <xcon:floor id="jSummitFloor">
                  <xcon:media-label>
                        jSummitLabel
                  </xcon:mediaLabel>
                </xcon:floor>
          </xcon:conference-floor-policy>
        </xcon:floor-information>
      </blueprintInfo>
    </ccmp:blueprintResponse>
  </ccmpResponse>
</ccmp:ccmpResponse>
```

As a reply to a 'blueprintRequest' message, the CCMP response includes a 'blueprintResponse' specialized message in its body: this element includes the whole conference object (compliant with the XCON common data model specification) associated with the requested blueprint, as part of a <blueprintInfo> container. Besides containing some of the parameters provided in the request (confUserID, confObjID, operation), the response also carries back an additional piece of information related to the result of the request, namely, a 'response-code' parameter telling the participant that the request was successfully taken care of ('200'), which is also reflected in the related 'response-string' ('Success').

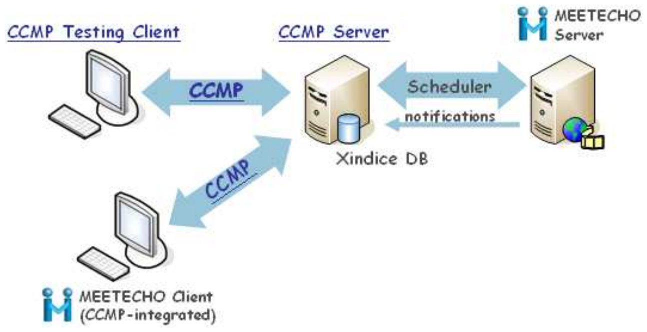


Figure 6: Reference scenario @ unina

The next section will provide further details on our implementation experience with the protocol. Specifically, we will address the way we designed the process according to the specification (both from the client and the server perspective), and the related implementation choices.

6. CCMP WORK AT UNINA

This section deals with our prototype implementation of the CCMP protocol. The reference scenario is the one depicted in Fig. 6.

As the figure shows, in order to have a working instance of the CCMP protocol which could be used as a playground for testing and validation of the specification in progress, as a first step we have realized a stand-alone Java-based CCMP client and a Java-based CCMP server.

The CCMP testing client presents a very simple graphical user interface through which it is possible to create and send to the CCMP Server the desired CCMP request. All CCMP messages sent and received by the client are logged onto a debugging window which allows to easily visualize the entire call flow associated with client-server interactions.

As to the server, it has been integrated into our Meetecho conferencing platform [1]. Since Meetecho already makes use of a "proprietary" protocol[1] for conference creation, manipulation and scheduling (which is herein called 'Scheduler'), we had to implement the CCMP server as a proxy towards it. The CCMP server receives CCMP requests from the testing client, converts them into Scheduler requests and forwards them to the Meetecho server by using the Meetecho Scheduler protocol, which is a simple, text-based protocol based on TCP. When the Meetecho server is done with the forwarded request, it sends back to the CCMP server a Scheduler-compliant answer, which is then converted into a CCMP-compliant response and forwarded to the CCMP testing client. The CCMP server takes care of the correct mapping between CCMP- and Scheduler-compliant messages. We also remark that synchronization between the Meetecho server and the CCMP proxy server can be achieved through asynchronous notifications. As soon as something worth communicating happens at the Meetecho server, a notification can be sent to the CCMP proxy (which subscribes to the events associated with conference management and manipulation) in order to let it always have an up-to-date view of the actual situation inside the conferencing server.

[1]This is due to the fact that, when the Meetecho XCON-compliant conferencing platform has been conceived, inside the XCON Working Group there was no consensus yet as to the standard conference control protocol to be adopted.

Indeed, the notification mechanism described above, allows us to improve the overall performance of the integrated server made of the CCMP proxy combined with the Meetecho server. In fact, provided that the CCMP proxy is always kept aligned with the Meetecho server for all what concerns conference-related information, we can let it respond to CCMP client requests directly, thus skipping the complex operations associated with the needed 'CCMP-Scheduler' mapping procedures, along both directions.

Upon activation, the CCMP Server retrieves, through specific Scheduler requests, all the Meetecho blueprints and conferences and loads them into a native XML database. Conference objects hence take the form of XML conference documents compliant with the XCON data model. As stated above, the CCMP Server is also a subscriber to the Meetecho Notification Service, and is thus aware of all modifications taking place on the conferences managed by the Meetecho server (modifications which might also be due to actions undertaken by non-CCMP aware Meetecho conferencing clients). Accordingly to the received notifications, the database is updated. In such a way, the CCMP Server has always available an aligned image of the conference information set managed by the Meetecho platform. This allows the CCMP Server to immediately answer to CCMP retrieve requests, without forwarding the corresponding Scheduler request to the Meetecho server each time this kind of message arrives. Unlike the retrieve case, the CCMP requests associated with an operation of either create, or update, or delete must be translated into the equivalent Scheduler messages to be sent to Meetecho, in order to have an actual effect on the Meetecho Server side. The Scheduler responses are then interpreted and converted into the appropriate database updates, as well as translated into the equivalent CCMP responses to be returned to the CCMP Client.

Having transposed the Meetecho Conference Control plane to the CCMP world, we have integrated a library of CCMP APIs into our Meetecho client, thus allowing it to make use of CCMP (instead of the legacy Meetecho Scheduler protocol) as the Conference Control Protocol, in such way completing the scenario we presented in Fig. 6.

In the following subsections, we delve somehow into the details of the main actors involved in the Conferencing Control environment we have introduced, namely the CCMP client, the CCMP proxy server and the native XML database. We will discuss both the design and the implementation choices associated with the above mentioned components. Some notes and considerations about the way CCMP has been integrated into the Meetecho client are also reported.

6.1 Managing XML CCMP messages and conference information

The CCMP server and client components have both been implemented in Java. Since CCMP messages, as well as the conference-related information they carry, are formatted as XML documents, we faced the need of generating, parsing and handling such items in Java. Besides, as it will be explained in the following section, we also needed a proper way to handle an XML-aware database, which could manage the manipulation of conference objects.

In order to facilitate these operations, we chose to exploit the JAXB API (Java Architecture for XML Binding API)

2.1, which is the last API version at the time of this writing. This API allows to represent XML documents (that also have to be validated against a given XML Schema) in a Java format, i.e. through Java objects representing their different composing parts. The binding indeed represents the correspondence between XML document elements and the Java objects created with JAXB. Accessing XML contents by means of JAXB presents several advantages in terms of both efficiency and easiness with respect to SAX and DOM parsing. In fact, just like DOM, the output analysis can be saved at once and then consulted at any time without having to re-parse the whole document again, while the concerning memory occupation turns out to be lower than the one of the DOM tree; like SAX, on the other hand, it is possible to access specific document parts without performing a further complete document parsing and without traversing the XML tree until the leaf to be examined is reached.

JAXB not only allows for easy access to XML documents, but also for a seamless creation of XML documents from the representative Java counterparts. This operation is called 'marshalling'. The inverse operation, from XML to Java objects, is instead called 'unmarshalling'.

Each JAXB generated class, corresponding to a specific type of XML element or attribute described in the schema file, is equipped with get and set methods that make it very easy to both extract information values and set them.

The JAXB architecture is composed of a set of APIs (contained in the `javax.xml.bind` extension package) and of a binding compiler, called XJC, which generates, starting from an XML Schema, the set of Java classes representing the element types embedded in XML documents compliant with it. In this context we have used the XJC Eclipse plug-in and produced the package of Java classes related to the XML Schema files collected from the data model documents [8, 7], as well as from the most up-to-date CCMP draft [3].

6.2 Managing HTTP

Considering the suggested transport for CCMP messages is HTTP (precisely, POST and 200 messages for requests and responses, respectively), we also had to cope with the issue of handling HTTP messages both at the client and at the server sides.

For the CCMP server implementation, we made use of the Apache open-source servlet engine Tomcat. The CCMP server business logic is realized through a servlet which, in the `doPost()` method, extracts the CCMP body from the HTTP POST request and, once the proper CCMP request type has been detected, starts the specific management thread accordingly.

On the client side, instead, we made use of the HTTP open source package provided by Apache, Apache Commons HTTP Client 3.1. This package is widely deployed in several projects, and allowed us to easily create and send to the CCMP server HTTP POST requests containing the CCMP message inside their payload, as well as handle the associated HTTP response accordingly.

6.3 Xindice database

We previously mentioned the need for an XML-aware database. In fact, CCMP handles the manipulation of conference objects compliant with the XCON common data model specification. Such conference objects are XML documents, and so, having an XML-aware database to store and manipulate them instead of relying on a relational databases relieved us form the burden of taking care of the transformation from tables to XML documents and viceversa whenever needed.

To cope with this requirement we chose Xindice, an open source Apache server handling an XML-native database specifically conceived for storing XML documents. Just as we needed, it allows to simply insert the XML data as it is when writing to the database, as well as to return the data in the same format when accessing the database. This feature is very useful when having to deal with complex XML documents like XCON conference objects, which might become very difficult or even impossible to be effectively stored in structured databases.

Xindice is installed as a Tomcat web application, and as such it was seamlessly integrated into our CCMP server prototype implementation. The `XML:DB` Java API is used to access the XML database. Such API is vendor-neutral, meaning that they are independent of the specific native XML database implementation, and operate on XML document collections, allowing the user to perform, on the collected XML documents, XPath queries as well as XUpdate modifications. Document collections are created and accessed through Xindice-specific Java APIs (Xindice Collection Manager Service).

In this context, we generated two main collections: (i) *confs* – the set of active and registered conferences hosted on the Meetecho server, reported in the form of conference documents compliant with the XCON data model; (ii) *blueprints* - the set of the Meetecho conference templates, in the XML XCON data model compliant format as well. A snapshot of the database content is showed in Fig. 7.

XPath queries are then executed by the CCMP server whenever needed, for instance to select the conference object referred to by the confObjID in CCMP requests, or to retrieve specific conference information from the XML conference documents grouped in the database collections.

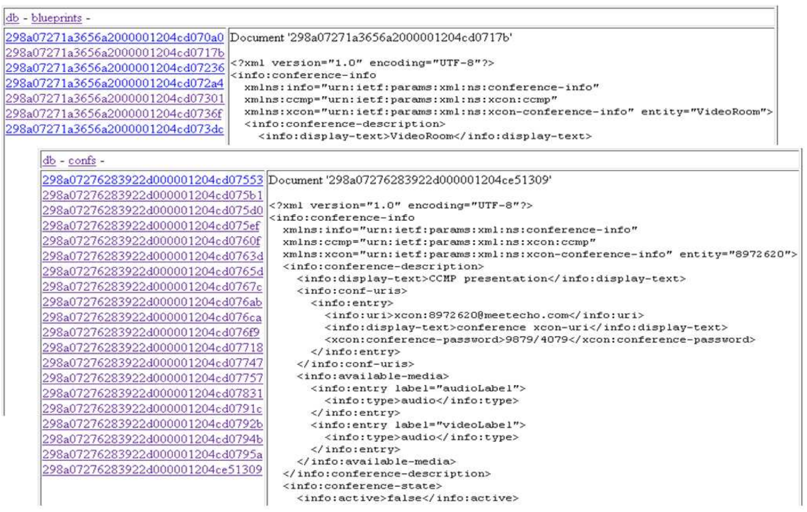
XUpdate queries are instead performed to update the conference documents according to received Meetecho notifications (generated, for example, as a consequence of a new user join or leave event) and CCMP client requests (e.g. when a client sets via CCMP a participant as chair of a certain floor).

6.4 CCMP-Meetecho integration

As anticipated, our reference conferencing platform, Meetecho, does not support CCMP natively. It instead currently relies on a proprietary protocol, called Scheduler, to handle conference objects and their manipulation. This protocol has a limited set of functionality available, which nevertheless can be logically mapped in a quite straightforward way to a subset of CCMP operations. This motivated us into integrating CCMP in our platform by handling at first CCMP as a simple wrapper to the operations made available by the Scheduler. This mapping is presented in Fig. 8.

Specifically, the Scheduler protocol allows a participant to: (i) create a new conference; (ii) delete existing conferences; (iii) retrieve the list of available blueprints; (iv) retrieve a specific blueprint; (v) setting a participant as floor chair of a media; (vi) retrieving the list of users in a conference. All these operations are made available by CCMP as well, and so this allowed us to test our prototype CCMP implementation in realistic scenarios.

The integration was realized by implementing a wrapper



db - blueprints -

298a07271a3656a2000001204cd070a0	Document '298a07271a3656a2000001204cd0717b'
298a07271a3656a2000001204cd0717b	
298a07271a3656a2000001204cd07236	
298a07271a3656a2000001204cd072a4	
298a07271a3656a2000001204cd07301	
298a07271a3656a2000001204cd0736f	
298a07271a3656a2000001204cd073dc	

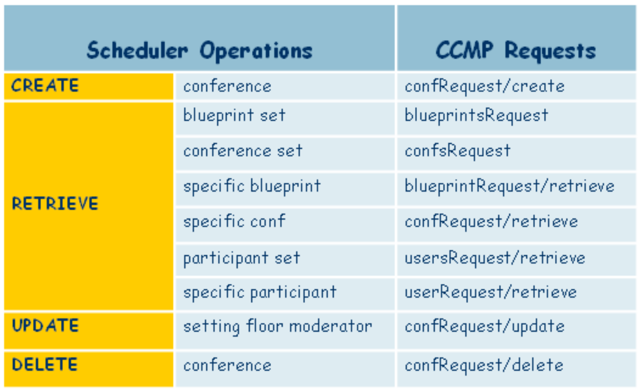
```
<?xml version="1.0" encoding="UTF-8"?>
<info:conference-info
  xmlns:info="urn:ietf:params:xml:ns:conference-info"
  xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
  xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info" entity="VideoRoom">
  <info:conference-description>
    <info:display-text>VideoRoom</info:display-text>
```

db - confs -

298a07276283922d000001204cd07553	Document '298a07276283922d000001204ce51309'
298a07276283922d000001204cd075b1	
298a07276283922d000001204cd075d0	
298a07276283922d000001204cd075ef	
298a07276283922d000001204cd0760f	
298a07276283922d000001204cd0763d	
298a07276283922d000001204cd0765d	
298a07276283922d000001204cd0767c	
298a07276283922d000001204cd076ab	
298a07276283922d000001204cd076ca	
298a07276283922d000001204cd076f9	
298a07276283922d000001204cd07718	
298a07276283922d000001204cd07747	
298a07276283922d000001204cd07757	
298a07276283922d000001204cd07831	
298a07276283922d000001204cd0791c	
298a07276283922d000001204cd0792b	
298a07276283922d000001204cd0794b	
298a07276283922d000001204cd0795a	
298a07276283922d000001204ce51309	

```
<?xml version="1.0" encoding="UTF-8"?>
<info:conference-info
  xmlns:info="urn:ietf:params:xml:ns:conference-info"
  xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
  xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info" entity="8972620">
  <info:conference-description>
    <info:display-text>CCMP presentation</info:display-text>
    <info:conf-uris>
      <info:entry>
        <info:uri>xcon:8972620@meetecho.com</info:uri>
        <info:display-text>conference xcon-uri</info:display-text>
        <xcon:conference-password>9879/4079</xcon:conference-password>
      </info:entry>
    </info:conf-uris>
    <info:available-media>
      <info:entry label="audioLabel">
        <info:type>audio</info:type>
      </info:entry>
      <info:entry label="videoLabel">
        <info:type>audio</info:type>
      </info:entry>
    </info:available-media>
  </info:conference-description>
  <info:conference-state>
    <info:active>false</info:active>
```

Figure 7: An image of the Xindice native XML database used in the prototype



Scheduler Operations		CCMP Requests
CREATE	conference	confRequest/create
RETRIEVE	blueprint set	blueprintsRequest
	conference set	confsRequest
	specific blueprint	blueprintRequest/retrieve
	specific conf	confRequest/retrieve
	participant set	usersRequest/retrieve
	specific participant	userRequest/retrieve
UPDATE	setting floor moderator	confRequest/update
DELETE	conference	confRequest/delete

Figure 8: CCMP-Scheduler mapping

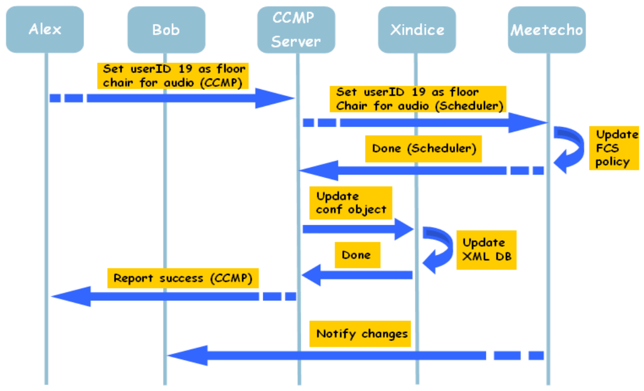


Figure 9: A sample CCMP-based interaction involving protocol mapping

on the server side. We deployed our CCMP server (Tomcat, JAXB and Xindice) by putting it side by side with the existing Meetecho server. We then added to the already implemented CCMP server logic a wrapping functionality, in order to handle incoming CCMP requests and translate them into Scheduler directives accordingly, where applicable, and viceversa. On the client side, we replaced the Scheduler client module with our CCMP client implementation and logic.

The mode of operation is quite straightforward. Any time a participant issues a CCMP request, it is handled by the CCMP server. The CCMP server maps the request to the Scheduler counterpart, translating the message. Such a message is then forwarded to the legacy Meetecho Scheduler server, where it is handled and enforced. According to the Scheduler reply that is received as a consequence, the CCMP server takes the related action, e.g. updating the XML conference object on the Xindice database if needed, and providing the participant with a coherent CCMP response.

An example is provided in Fig. 9.

A dump of the CCMP messages exchanged follows:

```
ccmpRequest message sent:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpRequest
      xmlns:info="urn:ietf:params:xml:ns:conference-info"
      xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp"
      xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info">
   <ccmpRequest
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ccmp:ccmp-conf-request-message-type">
      <confUserID>xcon-userid:alex@meetecho.com</confUserID>
      <confObjID>xcon:8977777@meetecho.com</confObjID>
      <operation>update</operation>
    <conference-password>1377</conference-password>
      <ccmp:confRequest>
         <confInfo entity="xcon:8977777@meetecho.com">
        <xcon:floor-information>
               <xcon:conference-floor-policy>
                <xcon:floor id="11">
            <xcon:moderator-id>19</xcon:moderator-id>
                </xcon:floor>
               </xcon:conference-floor-policy>
          </xcon:floor-information>
         </confInfo>
      </ccmp:confRequest>
   </ccmpRequest>
</ccmp:ccmpRequest>

ccmpResponse message received:

  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ccmp:ccmpResponse
       xmlns:xcon="urn:ietf:params:xml:ns:xcon-conference-info"
       xmlns:info="urn:ietf:params:xml:ns:conference-info"
       xmlns:ccmp="urn:ietf:params:xml:ns:xcon:ccmp">
    <ccmpResponse
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:type="ccmp:ccmp-conf-response-message-type">
       <confUserID>xcon-userid:alex@meetecho.com</confUserID>
       <confObjID>xcon:8977777@meetecho.com</confObjID>
       <operation>update</operation>
       <response-code>200</response-code>
      <response-string>Success</response-string>
      <version>10</version>
    <ccmp:confResponse/>
  </ccmpResponse>
</ccmp:ccmpResponse>
```

This simple example shows the process for a typical scenario. In an active conference (identified by its ID 8977777), a participant, Alex (who happens to be administrator of the conference), decides to assign a floor chair for the audio resource, in order to have it properly moderated by means of BFCP. In CCMP, this is achieved by issuing a 'confRequest' with an 'update' operation: the body of the specialized 'confRequest' element contains the part of the conference object that needs manipulation, in this case the floor information associated with the existing audio resource. This audio resource is identified by means of a floor id (11 in the example), which in the conference object itself is explicitly mapped to the label assigned to the audio medium. Since Alex is interested in assigning a floor chair to take care of this medium, he specifies a 'moderator-id' (19) which refers to a specific userID in the Floor Control Server. A password is also provided (1377) since this operation requires special permissions.

This request is sent to the CCMP server which, since a mapping with the Scheduler functionality exists, translates the message accordingly to the Scheduler format, and sends the newly created message to the legacy Meetecho Server. The server handles the request and enforces it, updating the Floor Control Server policy accordingly. The successful result of the operation is reported by means of a Scheduler reply to the CCMP server, which in turn updates the Xindice database coherently with the request. This means that the XML conference object associated with conference 8977777 is updated. A success is finally returned to the participant by means of a CCMP response.

7. CCMP HISTORY AND RELATED WORK

Every time a framework for conferencing has been proposed, the need for a proper Conference Control mechanism has arisen as a consequence. For this reason, such mechanism has been the subject of a lot of efforts. Nevertheless, the proprietary nature of most of the conferencing solutions currently available paved the way to numerous heterogeneous and incompatible solutions for such a functionality. For the sake of conciseness, we don't provide in this section a list of such solutions, considering it would be quite incomplete. We instead focus on the related work carried within the standardization bodies. In fact, since the XCON architecture has been introduced within the IETF, several different candidates have been proposed to play the role of the Conference Control Protocol. Such candidates differed in many aspects, which reflected the discussion within the standardization fora with respect to the approach that should be taken in that sense. An interesting debate took place, for instance, about whether a semantic or a syntactic approach would be better as a basis for a Conference Control Protocol. Besides, the best transport means to be adopted has also been the subject of investigation.

In this spirit, at least three candidates were proposed in the XCON WG before CCMP was chosen as the official protocol.

The first proposal, at the end of 2004, was the "Centralized Conference Control Protocol" (`draft-levin-xcon-cccp`) by O. Levin and G. Kimchi. This protocol, as CCMP, was XML-based and had a client-server organization, but unlike CCMP it was designed using SOAP as a reference model. It likely reflected the implementation work carried out within Microsoft at the time. Despite being in a quite advanced state (four updates were submitted), the proposal was eventually put aside.

Shortly after the first individual submission of the CCCP, another candidate came to the light, "COMP: Conference Object Manipulation Protocol" (`draft-schulzrinne-xcon-comp-00`) by H. Schulzrinne. Like its predecessors, it heavily relied on Web Services as a reference, while stressing the use of SIP for notification purposes. Unlike CCCP, COMP had a strong semantic approach for what concerned the protocol specification. No updated versions of the draft were submitted; this work nevertheless paved the ground to a stimulating discussion that eventually led to CCMP.

One month later, another candidate was proposed, the "Conference State Change Protocol (CSCP)" by C. Jennings and A. Roach. Unlike both its predecessors, CSCP took a completely different approach towards the protocol. In fact, CSCP was basically a proposal to extend the already defined Binary Floor Control Protocol in order to allow it to also deal with conference manipulation functionality. CSCP motivated such an approach stressing the fact that binary messages would be smaller and easier to handle, especially for mobile devices. Besides, it was the authors' opinion that every XCON-compliant entity would likely support BFCP already, and as such CSCP would prove a trivial addition. Nevertheless, the proposal was eventually abandoned, and a text-, possibly XML-based solution was decided to be a preferred approach.

Finally, a last proposal saw the light at the end of 2005, the individual submission that would subsequently become the official CCMP draft. Such a draft has seen many revisions and efforts since then, which have resulted in the work presented in this paper.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have presented the design and implementation of the Centralized Conferencing manipulation Protocol (CCMP), currently on the way towards its steady-state as a standard IETF protocol for conference objects management in the XCON framework.

We highlighted the main motivations behind such a work and illustrated the complex path that has been followed within the IETF community along the many phases of the overall standardization process.

We first described the general structure of the protocol, as well as its main functionality. Then, we focused on the work carried out at the University of Napoli during these last years and centered around a running prototype acting as a major playground for all the activities associated with ongoing standardization work inside some of the key working groups of the RAI (Real-time Applications and Infrastructure) area of the IETF.

At the time of this writing, the specification of the CCMP protocol is close to completion. Its implementation has been heavily used to both test its behavior and to provide invaluable feedback to the authors of the CCMP document. Furthermore, to aid implementors, a specific draft focusing on call flows has been written in order to provide the Internet community with guidelines in the form of Best Common Practices.

Our future work related to CCMP will definitely concern the final refinement of the specification, with the goal of arriving at a well-assessed RFC document. As to the implementation, we are currently working on the integration of the CCMP server within the Meetecho platform as a 'native' component, in such a way as to avoid the unavoidable burden associated with proxying CCMP requests and mapping them onto the legacy scheduler protocol.

When done with such integration, we will also focus on carrying out a thorough experimental campaign aimed at assessing the performance achievable by our protocol implementation, as well as identifying its potential bottlenecks.

9. REFERENCES

[1] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano. Meetecho: A standard multimedia conferencing architecture. In *FMN '09: Proceedings of the 2nd International Workshop on Future Multimedia Networking*, pages 218–223, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] M. Barnes, C. Boulton, and O. Levin. RFC 5239 - A Framework for Centralized Conferencing. Request for comments, IETF, June 2008.

[3] M. Barnes, C. Boulton, S. Romano, and H. Schulzrinne. Centralized Conferencing Manipulation Protocol (work in progress). Internet draft, IETF, June 2010.

[4] G. Camarillo, S. Srinivasan, R. Even, and J. Urpalainen. Conference event package data format extension for centralized conferencing (xcon) (work in progress). Internet draft, IETF, September 2008.

[5] Fielding. Architectural styles and the design of network-based software architectures. Technical report, 2000.

[6] M. Gudgin, N. Mendelsohn, M. Hadley, J. Moreau, and H. Nielsen. Soap version 1.2 part 1: Messaging framework. World wide web consortium first edition rec-soap12-part1-20030624, W3C, June 2003.

[7] O. Novo, G. Camarillo, D. Morgan, and J. Urpalainen. Conference information data model for centralized conferencing (xcon) (work in progress). Internet draft, IETF, February 2010.

[8] J. Rosenberg, H. Shulzrinne, and O. Levin. RFC 4575 - A Session Initiation Protocol (SIP) Event Package for Conference State. Request for comments, IETF, August 2006.