API for Shongo

Petr Holub, Jan Růžička, Miloš Liška, Martin Šrom, Ondřej Pavelka, Ondřej Bouda

Contents

Chapter 1

Use Cases

1.1 Resources

UC-1 (res:types) Types of resources

TODO: Describe managed/unmanaged

Basic Types:

- specific device/server
- specific physical/virtual room
- specific license
- specific identifier (Adobe Connect URL, H.323 number)

TODO: Describe each basic type

TODO: Will be identifier reservations done the same way as all other resources?

UC-2 (res:management) Management of resources

The resource owner should be able to create new resources that will be managed by Shongo. Owner should be able to modify the managed resource parameters and also should be able to delete the managed resource.

TODO: Will shongo know about unmanaged resources? Should the user fill all the necessary attributes every time he wants to add unmanaged resource to reservation?

UC-3 (res:identification:managed) Managed resource identification

Each managed resource is identified by an unique identifier. The identifier will be assigned to the resource by it's owner when the resource is being created (TODO: or modified too?). The identifier follows the URN standard (TODO: reference):

`urn:id:domain(.subdomain)*.name`

Examples:

- `urn:id:cz.muni.fi.sitola.c90` – H.323 endpoint at C90 room
- `urn:id:cz.cesnet.srom` – personal H.323 Mirial endpoint

Each managed resource has own **name** and belongs to some main **domain**. The domain then can be structured to any number of **subdomain**s.

By the resource identifier, the user can lookup resource type and all other attributes.

UC-4 (res:identification:unmanaged) Unmanaged resource identification

Each unmanaged resource is identified by an unique identifier that contains specified **type** (e.g., H.322 endpoint) and the resource **identity** for specified type (e.g., H.323 number):

`urn:id:unmanaged:type:identity`

Examples:

- `urn:id:unamanged:h323:950000001`
- `urn:id:unmanaged:connect:<shibboleth>`
  TODO: Is it possible?
  TODO: How large is shibboleth or other possible identity?

By the unmanaged resource identifier it is possible to determine resource type and concrete identity in that resource type.

1.2 Reservations

UC-5 (rsv:specifications) Types of specifications

Specification of a resource, being object of a reservation, may be of the following types:

- a *fully-qualified explicit specification (FQESpec)* – specifies exactly one element; it ma refer to a specific device (e.g., H.323 endpoint, web browser as an endpoint for Adobe Connect), a specific server (e.g., a specific Adobe Connect server or H.323 MCU), a specific physical room, or a specific virtual room (e.g., a specific room running on specific H.323 MCU),
  FQESpec may be managed by Shongo or not; for resources that Shongo does not manage or knows about, i.e., unmanaged resources, the user needs to specify type of the resource (e.g., generic H.323 endpoint). The unmanaged resources should have some form of identification (e.g., H.323 number, H.323 ID, or Shibboleth identity for Adobe Connect) so that Shongo can verify if they are connected to the virtual room or not during the conference.
  Anonymous unmanaged resources may also be available (completely generic H.323 enpoint without a number or H.323 ID, or guest user in Adobe Connect) , but some functionality may not be available – when maximum room capacity is achieved (or exceeded), anonymous users not be allowed in (or even be disconnected in LIFO mode until maximum amount of participants is obeyed).
- a *partially-qualified explicit specification (PQESpec)* – specifies a class/type of a resource (e.g., H.323 endpoint) and it is up to the scheduling to find suitable one (combination of availability and access-level for given user),
- a *implicit specification (ISpec)* – the user does not specify such a resource, but the resource is needed to implement user's request (e.g., if user specifies Connect and H.323 endpoints, a gateway/connector is needed to implement the translation; if user specifies multiple H.323 endpoints beyond MCU-capability of each of them, some MCU is needs to be included).

Generally, Shongo should use the technology to limit number of participants in the rooms created based on the reservations—e.g., H.323 MCUs allow for setting an upper limit on number of participant in each room.

UC-6 (rsv:roles) User roles

Each reservation should have at least two types of possible user roles:

- *owner/administrator*, who can modify or even delete the reservation,
- *manager*, who can control the room (e.g., disconnect participants, mute participants, etc.),
- *participant*, who can only view the reservation including coordinates necessary for participation.

The roles can be delegated, which is important especially in case of owner/administrator: the original reservation creator can delegate this role to other users and any of them can the modify or delete the reservation.

UC-7 (rsv:identification) Reservation identification

Each reservation is identified by unique identifier. The identifier follows the URN standard (`TODO: reference`) and it is assigned to reservation automatically:

```
urn:uuid:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

The **`xxx...`** represents UUID which is generated automatically and it provides uniqueness (`TODO: reference`).

UC-8 (rsv:reservation:one) One time reservation

Common type of reservation, where a user requests certain resources for limited time duration. Unlimited reservations are not assumed by this scenario (see UC-10).

Start time of a reservation may be any time in the future or *now*, which is also called *ad hoc* reservation.

Reserved resources may be given as FQESpec, PQESpec, or ISpec. FQESpec are either accepted or denied by the scheduler, while other types of the specifications are looked for their best match. PQESpec may include the following:

- user may request a general endpoint and Shongo should try to find the closest matching endpoint available to the user (e.g., user requests a H.323 endpoint for a conference since she has no personal endpoint, and she is assigned a room-based H.323 endpoint provided the room is available),

while examples of ISpec are as follows:

- amount of central resources (such as H.323 MCU ports or Connect licenses) based on specified number of (H.323/SIP or web-browser) participants,
- any interconnecting elements (e.g., gateways) to interconnect the endpoints specified by the user; if only part of the endpoints can be interconnected, the user should be notified what parts can be interconnected and what parts are disconnected.

Each reservation has to be given a unique identifier that is further used for any references to it. If the reservation is denied, reasons for denying should be communicated to the requester. In case that the reservation succeeds, all the users involved should be notified.

Each reservation has to include:

- unique identifier,
- timespan definition,
- requester's identifier,
- name,
- links to the resources involved, including specification of the amount of resources consumed,
- list of users involved.

UC-9 (rsv:reservation:periodic) Periodic reservation

UC-8 extended with periodicity. Expressiveness of the periodicity language should be equivalent to cron plus start time, stop time or number of repetition, and explicit lists for recurring aperiodic requests.

UC-10 (rsv:reservation:permanent) Permanent reservation

This is specific type of reservation that can be only made by an owner of the resource as it permanently removes the reserved capacity from the dynamic Shongo scheduling.

Even permanent reservations must not threaten what has already been reserved for any user. In case of priority requests (see UC-11), Shongo must be able to migrate the reservation to other resources.

UC-11 (rsv:priority) Priority reservations

Priority reservations are only allowed by an owner of the resources and they may affect reservations already present on the resources. However, priority reservation should only be allowed if there is some other resource(s) (maybe even in another domain) that can take over the prior reservation. In case of reservation migration, all the involved users must be notified (see UC-17).

`TODO: We need to decide, whether to allow this or not.`

UC-12 (rsv:max-future) Maximum future time for reservations

Each resource owner should set a date/time limit in the future (e.g., 2 months), above which reservations are not allowed. That should be done for each owned resource. Whole reservation duration must fit in that limit. This limit ensures there is some time point in the future, where there are no reservations on the resource—e.g., for maintainance purposes, removal of the device, special events the device will be used for, etc.

UC-13 (rsv:lookup:time) Lookup available time

User may look up available time slots for given amount of requested resources, with either inter-domain negotiation turned off or on (i.e., tell the user when resources are available within the domain or when merging resources of all the domains).

UC-14 (rsv:list) List all the reservations

Some querying/filtering language needs to be supported to limit list to

- room types (H.323, SIP, Connect, etc.),
- equipment (be it class of equipment or a specific device).
- reservation owner(s),
- users involved (may be humans as well as resources, such as rooms with equipment) involved in the room as participants.

UC-15 (rsv:modify) Modification of a reservation

Any attribute of a reservation may be requested to change. The request may be accepted or denied by the scheduler. In case of the denial, reasons for denial should be communicated to the requester. If the modification succeeds, all the users involved should be notified.

UC-16 (rsv:release) Release/canceling of a reservation

All the users involved should be notified.

UC-17 (rsv:migration) Migration of a reservation

If the change is visible to the users (e.g., typically this would include change of the server/MCU the users connect to), all the users involved should be notified.

UC-18 (rsv:notification) Notification of participants

In case of making, modifying, or canceling a reservation, all the users involved should be notified, as specified in UC-8, UC-9, UC-10, UC-15, UC-16, and UC-17. By default, the users should be notified via email, but it would be interesting to provide also SMS notification service.

UC-19 (rsv:service-users) Reservations of rooms, public or semi-private endpoints, etc.

Each reservation may include endpoint resources (beyond human users with private endpoints—H.323/SIP/web), which represent entities such as rooms, non-personal endpoints, etc., that can be scheduled in a similar way to central resources.

This type of reservation may be either part of some infrastructure reservation (see UC-8, UC-9, UC-10) or standalone reservation (e.g., reservation of a meeting room with H.323 equipment to disable the room from scheduling for given time duration).

UC-20 (rsv:recording) Reservation of recording capacity

Usually part of some infrastructure reservation (see UC-8, UC-9, UC-10), but may be completely standalone in case that only recording server is used of the Shongo-managed infrastructure.

UC-21 (rsv:streaming) Reservation of streaming capacity

May part of some infrastructure reservation (see UC-8, UC-9, UC-10), but may be completely standalone in case that only streaming server is used of the Shongo-managed infrastructure.

1.3 Operations

UC-22 (ops:migration) Live migration of a virtual room

This use case is intended for migration due to planned server maintenance or unplanned server outage. Ideally, all the room settings and content should be transferred to the target room—but some content may be lost in case of unplanned server failure (namely content migration).

Being able to transfer room settings to another server in case of unplanned failure also requires that the settings needs to be stored in the Shongo middleware.

Clients should be automatically redirected to the new server, if technology permits, or at least notified of the migration (email, SMS—see UC-18).

Some functionality will be common UC-17.

1.3.1 Room Management

UC-23 (ops:room:shongo-options) Get room information on Shongo level

This information typically includes name, owner, date/periodicity, duration and type.

UC-24 (ops:room:users-list) List users

Each user should be given a unique identifier in the output list that can be used for further querying. It should also provide means to identify the same user (e.g., if the user disconnects–reconnects, it should contain a part that is common and that denotes the specific user and a part that is specific for the session, so that if the user is connected twice (one session is in timeout state and the other session has just been established), we can differentiate between the two sessions).

UC-25 (ops:room:user-info) Print detailed info about a user in a room

Print all the statistics we can get about a user participating in the room. It should contain technology agnostic part (e.g., when the user joined) and technology specific part (i.e., H.323 statistics, H.245/SIP capabilities negotiation info, H.239 content information, etc.).

UC-26 (ops:room:layout) Set room layout

Shongo should be able to set up global layout of a room and user-specific layout, if available through API of virtual room provider.

UC-27 (ops:room:user-disconnect) Disconnect a user

Immediate disconnection of a user.

UC-28 (ops:room:disable-user-content) Disable H.239 content from a specific user

Disable content the user to be H.239 content provider for the given room.

UC-29 (ops:room:specific-user-content) Enable H.239 content only from a specific user

Enable H.239 content only from the specific user, typically by disabling content from all other users. Normally, users may fight who is going to be the content provider.

UC-30 (ops:room:user-mute) Mute a user

Mutes user on the room level. Optionally if user's endpoint is also controlled by Shongo, it should provide means to mute the endpoint (which can be easily unmuted by the user).

UC-31 (ops:room:user-miclevel) Set microphone audio level for a user

Sets the audio from the user on the room level. Optionally, if user's endpoint is also controlled by Shongo, it should provide means to control mic level on the endpoint. In this case, audio should be normalized on the endpoint before doing modifications on room level (if the sound is too low or too high and distorted, it may not be corrected on the MCU).

UC-32 (ops:room:user-playlevel) Set playback audio level for a user

This functionality is typically available only when user's endpoint is also controlled by Shongo.

UC-33 (ops:room:user-video-off) Disable video of a user

UC-34 (ops:room:user-video-snap) Video snapshot for a user

If provided by the room provider (MCU, web conferencing, etc.), we should be able to get video snapshot of:

- video sent by the user,
- video received by the user.

UC-35 (ops:room:user-layout) Set layout specific for a user

UC-36 (ops:room:settings-down-up) Download and upload room settings

We should provide an API that allows for downloading settings of the room to the maximum extent possible, in order to back it up and reupload it later on. This is a convenient way to back up setting as well as to reset a newly created room (e.g., as a part of a new reservation) to old settings.

UC-37 (ops:room:content-down-up) Download and upload room content (if technology permits)

If technology and access policy permits, we should be able to download and upload content of the room (e.g., documents, notes, polls, etc.). See UC-36.

UC-38 (ops:room:room-techspec) Get/set technology-specific properties for a room

This may include specific attributes of the room (typically on room provider level), such as enabled codecs.

UC-39 (ops:room:user-techspec) Get/set technology-specific properties for a user

UC-40 (ops:recordings-management) Management of recording archives

It should be possible to work with the recorded video through Shongo, e.g., migrate it from a content server to a storage of a streaming server. Plus it should be possible for owner/administrator or manager to access URLs of the recorded content to send them via email. Also, it should be possible to automatically notify all the (non-anonymous) participants about the recording via email.

1.4 Monitoring & Management

1.4.1 Shongo management and monitoring

UC-41 (mgmt:shng:list-agents) List of all the agents in the system

The listing API must including querying language that allows selection of only a subset based on similar properties like those defined in UC-14.

UC-42 (mgmt:shng:list-controllers) List primary and backup controllers

List all the controllers (primary and backup) for current domain.

UC-43 (mgmt:shng:list-domains) List domains

List of all other known domains including references to their domain controllers and state of connections to them.

1.4.2 Server management and monitoring

UC-44 (mgmt:srv:get-load) Get server load

The API should provide means to get load on the server machine, containing at least the following:

- CPU load
- memory load

- disk occupancy

Obviously, this information may or may not be available for specific device. In case that the information is not available, the API should report this in a consistent way (specific exception or unique return value).

UC-45 (mgmt:srv:schedule-downtime) Schedule server downtime

Downtime scheduling must include change/migration of all the reservations and live events influenced by the downtime. Conceptually, this is similar to permanent reservations a bit (UC-10)—the major difference is that during the downtime, the resource is not available to Shongo for management and this state is intentional. Downtime is also per-resource and does not have participants.

UC-46 (mgmt:export-stats) Export Shongo stats

Export reservation stats in some common format like CDR.

Chapter 2

Common Data Types and Object Classes

2.1 Failure Related

TODO: : We need to decide what to do in case of failure. XMLRPC only allows **faultCode** and **faultString** to be returned as a kind of exception. If we find it insufficient, we'd probably need to look beyond XMLRPC.

2.2 Common

- **Attributes = Map<String, Object>**

  The **Attributes** type is used in API methods that perform modification of entities to allow user to specify only the attributes which he aims to update and not to force him to enumerate all attributes.

  TODO: Verify that it can be implemented in XML-RPC/SOAP

2.3 Security and Identity Related

- **SecurityToken**

  Contains identity and credentials of a user performing the requested operation.

2.4 Time Related

- **Date**

  Represented as ISO8601 date/time (e.g., 20120130T10:09:55). More efficient way of implementation may be used internally, of course.

- **PeriodicDate**

Extends **Date** type for periodicity. The extension is meant in an OOP manner, thus wherever **Date** is used as an argument in the API definition, the **PeriodicDate** instance can be passed as the argument too.

- **Duration**

  Represented as ISO8601 duration (e.g., P3Y6M4DT12H30M5S, which is *3 years, 6 months, 4 days, 12 hours, 30 minutes, and 5 seconds*).

- **TimeSlot**

  Time slot is pair of **Date** and **Duration**.

2.5 Reservations and Resources

- **ResourceType**

  Represents types (may be also understood as capabilities) of a resource. It is composed of:
  - technology type (**H323**, **SIP**, **AdobeConnect**)
  - resource role (**EndPoint**, **Server**)

  If resource role is **Server**, additional options become available:
  - server role (**MultiPoint**, **Gateway**)
  - translation type if server role is **Gateway** (subset of Cartesian product of technology types)

- **Resource**

  The resource may be of the following subtypes:
  - specific endpoint representation
    Refers to specific endpoint known to and managed by Shongo.
  - specific endpoint representation
    Generic endpoint of certain technology, not known or managed by Shongo.
  - licence representation
    Represents only pure licenses of certain technology on specific server or gateway.

- **ResourceSpecification**

  Represents resource specification and can be one of the types defined in UC-5. The specification instance should hold the right parameters for the chosen specification type.
  - for **FullExplicit** the parameter is concrete **Resource**.
  - for **PartialExplicit** the parameter is **ResourceType** (`TODO: or list of types?`).
  - for **Implicit** `TODO: some parameters?`.

  `TODO: How to implement it better? By three more types extending the base?`

- **ReservationType**

  The reservation may be of the following types:
  - **OneTime** One time reservation as defined in UC-8.
  - **Periodic** Periodic reservation as defined in UC-9.
  - **Permanent** Permanent reservation as defined in UC-10.

- **Reservation**

  It contains links to the following:

  - list of resources reserved directly as a part of reservation,
  - list of child reservations.

- **`Room`**

  Represents a virtual room on a specific resource of **`Server`** type (such as H.323 MCU).

- **`User`**

  Represents an active user in a **`Room`** on a **`Server`**. Ideally, it should also contain link to the user's eduID identity (this may or may not be available, though).

Chapter 3

User Interface API Specification

TODO: How to report failures? All by exceptions or some by return value?
=> exceptions
TODO: How to perform modifications? There are two possibilities:
1) by attributes - modifyEntity(Entity, Attr1, Attr2, ...)
2) by entity - modifyEntity(Entity), where Entity {Attr1, Attr2, ...}
=> option 1) but all attributes are merged into Attributes map

3.1 Resources

- **String createResource(SecurityToken token, String domain, String name, ResourceType type, String description)**

  Create a new resource that will be managed by Shongo. The user with given **token** will be the resource owner. The resulting value is the new resource unique identifier.

- **modifyResource(SecurityToken token, String resourceId, Attributes attributes)**

  Modify the resource with specified **resourceId** identifier. That operation is permited only when the user with given **token** is the resource owner.

  Possible Attributes:

  - TODO: Enumerate all possible attributes

- **deleteResource(SecurityToken token, String resourceId)**

  Delete the resource with specified **resourceId** identifier from Shongo management. That operation is permited only when the user with given **token** is the resource owner and only when the resource is not used in any future reservation.

- **setResourceMaximumFuture(SecurityToken token, String resourceId, Duration duration)**

  Set the maximum future time for reservations of resource with specified **resourceId** identifier that a user with given **token** owns.
  TODO: Keep this method or provide this feature by modifyResource?
  TODO: Do we want to have ability to disable resource for future reservations?

- **Resources[] listResources(SecurityToken token)**

  List of resources managed by Shongo, that a user with given **token** is entitled to see.

3.2 Reservations

TODO: We have agreed that when the user has a reservation, he should be able to request the same resources for another date/time. Example: He has periodical reservation on Mondays at three o'clock for some resources and exceptionally he wants the same reservation this Thursday. How to perform that?
1) Create new reservation with specified resources (this may reserve other resources in case of PQESpec and ISpec)
2) Each reservation can occur multiple date/times

- **`String createReservation(SecurityToken token, ReservationType type, Date date, Duration duration, String description, ResourceSpecification[] resources, String[] reservations)`**

  Create a new reservation of specified **`type`** at selected **`date`** for specified **`resources`**.

- **`modifyReservation(SecurityToken token, String reservationId)`**

  Modify the specified **`reservation`**.
  TODO: What changes will be able to apply?

- **`deleteReservation(SecurityToken token, String reservationId)`**

  Release the specified **`reservation`** and all children reservations.

- **`Reservation[] listReservations(SecurityToken token)`**

  List all the reservations that a user with given **`token`** is entitled to see.
  TODO: Use some querying/filtering language
  TODO: In which form should be Reservation returned?
  -XML-RPC can't return arbitrary objects, but only basic types and (recursive) Maps and Lists. Apache XML-RPC can be extended in the way to return Map representation of every object (that has tree structure, what about cycles?).
  -SOAP is able to return arbitrary objects.

- **`TimeSlot[] findReservationAvailableTime(SecurityToken token, ResourceSpecification[] resources, boolean interDomain)`**

  Lookup available time slots for specified **`resources`**. There is flag that specifies whether inter-domain lookup should be performed.

3.3 Room Operations

- **`Room[] listReservationRooms(SecurityToken token, Reservation reservation)`**

  Lists all the virtual rooms including rooms that are part of any child reservations.

- **`boolean isRoomActive(SecurityToken token, Room room)`**
- **`Resource[] listRoomResources(SecurityToken token, Room room)`**
- **`User[] listRoomUsers(SecurityToken token, Room room)`**
- **`disconnectRoomUser(SecurityToken token, Room room, User user)`**
- **`muteRoomUser(SecurityToken token, Room room, User user)`**

- setRoomUserMicLevel(SecurityToken token, Room room, User user, int level)
- setRoomUserPlaybackLevel(SecurityToken token, Room room, User user, int level)

Chapter 4

Connector API Specification

4.1 Data Types

4.2 Reservation API

4.3 User Management API

4.4 Monitoring API

4.5 Application Specific API

Chapter 5

Inter-Controller API Specification