International Standard

ISO/IEC 23093-5

First edition
2025-03

Information technology — Internet of media things —

Part 5: IoMT autonomous collaboration

Technologies de l'information — Internet des objets media —
Partie 5: Collaboration autonome dans l'IoMT

Reference number
ISO/IEC 23093-5:2025(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 23093 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The ISO/IEC 23093 series provides an architecture and specifies APIs and compressed representation of data flowing between media things (MThings).

The APIs for the MThings facilitate discovering other MThings in the network, connecting and efficiently exchanging data between MThings. The APIs also support transaction tokens to access valuable functionalities, resources, and data from MThings.

MThing-related information comprises characteristics and discovery data, mission descriptions from system designers and end-users, raw and processed sensed data and actuation information. The ISO/IEC 23093 series specifies input and output data formats for media sensors, actuators, storages, and analysers. In addition, media analysers can process sensed data from media sensors to produce analysed data, which can be cascaded to other media analysers to extract semantic information. Multiple MThings can be gathered and operated autonomously using mission descriptions given by system designers and end-users.

This document contains data formats and APIs to complete missions from system managers and end-users to operate multiple MThings autonomously. Refer to Figure 1 (items 1 and 1').

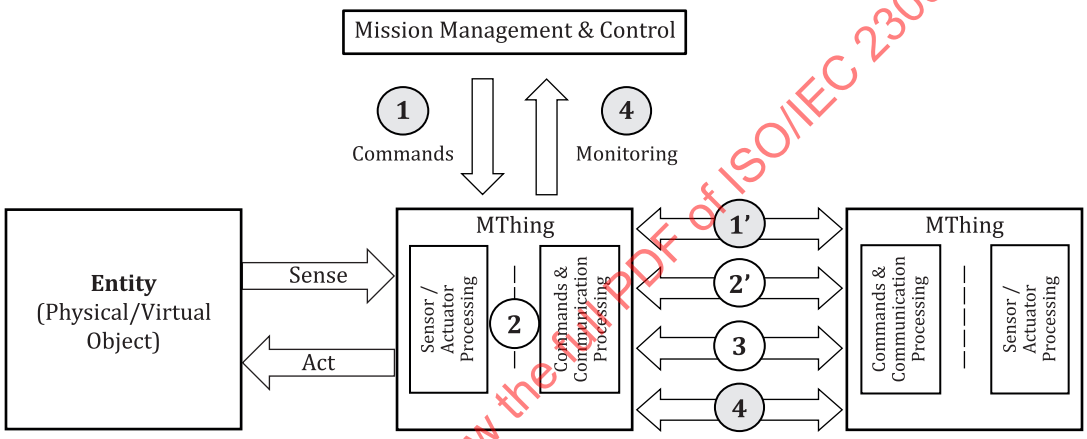

Figure 1 — Architectural view of the IoMT

Information technology — Internet of media things —

Part 5:
IoMT autonomous collaboration

1 Scope

This document specifies data formats and APIs for the mission management and control between MThings and end-users/system managers. Specifically, the following interfaces, protocols and associated media-related information representations are within the scope of this document:

— structured data formats (XML) representing the mission assigned by the user to the network of IoMT, for the data formats;

— structured data formats (XML) representing user commands to one or several MThings, possibly in a modified form (e.g. a subset of 1);

— APIs to exchange the data for mission management and control.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes the requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23093-1, *Information technology — Internet of media things— Part 1: Architecture*

ISO/IEC 23093-2, *Information technology — Internet of media things — Part 2: Discovery and communication API*

ISO/IEC 23093-3, *Information technology — Internet of media things — Part 3: Media data formats and APIs*

ISO/IEC 23093-6, *Information technology — Internet of media things — Part 6: Media data formats and APIs for distributed AI processing*

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23093-1, ISO/IEC 23093-2, ISO/IEC 23093-3, ISO/IEC 23093-6 and the following apply.

ISO and IEC maintain terminology databases for use in standardisation at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1
media analyser
MAnalyser
MThing (3.1.5) that can analyse media or metadata and produce interpreted media, metadata, or commands

3.1.2
media manager
MManager
MThing (3.1.5) that can register a list of MThings or be facilitated to search other MThings

3.1.3
media sensor
MSensor
MThing (3.1.5) that can sense and produce media data

3.1.4
media storage
MStorage
MThing (3.1.5) that can save media or metadata

3.1.5
media thing
MThing
thing (3.2.20 in ISO/IEC 23093-1:2022) capable of sensing, acquiring, actuating, or processing of media or metadata

3.1.6
media thing controller
MThing controller
MController
MThing (3.1.5) that can generate standard mission descriptions, control and monitor the progress of the mission of the participating *MThings* (3.1.5)

3.1.7
mission
task that *MThings* (3.1.5) shall carry out

3.2 Abbreviated terms

MCOV media thing controller output vocabulary

4 Schema documents

4.1 General

In the main text of this document, the syntax and semantics of data interfacing MThings are provided whenever possible as a single schema document.

In some cases, though, particularly for Clause 7, the syntax of data interfacing MThings is provided as a collection of schema snippets imbricated with other text. To form a valid schema document, users can gather these schema components in the same document with the schema wrapper provided at the head of the clause. For better readability, the relevant schema documents are available at https://standards.iso.org/iso-iec/23093/-5/ed-1/en/.

In all cases, each schema document has a `version` attribute, "ISO/IEC 23093-5." Furthermore, an informative identifier is given as the value of the `id` attribute of the `schema` component. This identifier is non-normative and used as a convention in this document to reference another schema document. In particular, it is used for the `schemaLocation` attribute of the `include` and `import` schema components.

4.2 Use of prefixes

For clarity, throughout this document, consistent namespace prefixes are used.

"xsi:" prefix is not normative. It is a naming convention in this document to refer to an element of the https://www.w3.org/2001/XMLSchema-instance namespace.

"xml:" and "xmlns:" are normative prefixes defined in Reference [1]. The prefix "xml:" is bound to "https://www.w3.org/XML/1998/namespace." The prefix "xmlns:" is used only for namespace bindings and is not itself bound to any namespace name.

All other prefixes used in either the text or examples of this document are not normative, e.g. "mpeg7:", “mcov:”.

In particular, most informative examples in this document are provided as XML fragments without the typically required XML document declaration and, thus, miss a correct namespace binding context declaration. The different prefixes are bound to the namespaces in these description fragments, as given in Table 1.

Table 1 — Mapping of prefixes to namespaces in examples and text

Prefix	Corresponding namespace
mpeg7	urn:mpeg:mpeg7:schema:2004
mcov	urn:mpeg:mpeg-IoMT:2024:01-MCOV-NS
xsi	https://www.w3.org/2001/XMLSchema-instance
xsd	https://www.w3.org/2001/XMLSchema

5 An overview of the mission management and control-related issues

5.1 Mission data usage process

5.1.1 General

This subclause describes the processes and examples of how the mission data are generated, transmitted, modified, and utilized to accomplish automatic collaborations among MThings.

5.1.2 Mission data usage scenario

5.1.2.1 Mission composition and transmission

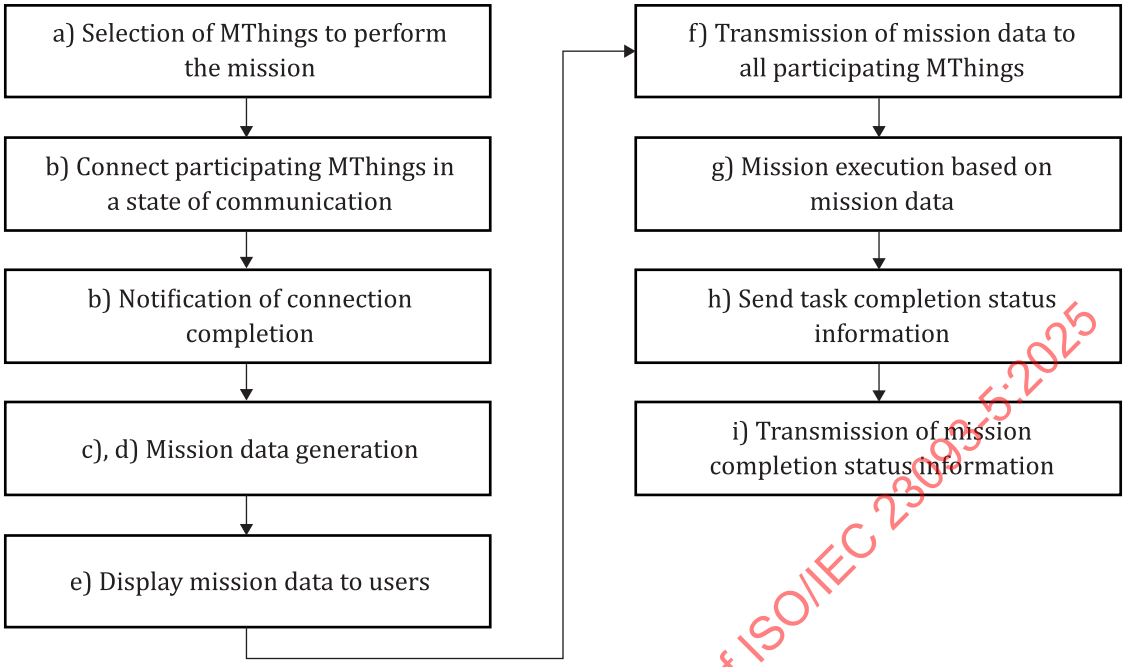

Figure 2 — Process of mission data composition and propagation

Figure 2 exemplifies how the mission data is generated, transmitted, and finalised. The process of composing and propagating the task descriptions of MThings could be as follows:

a) the user selects other MThings to complete the task desired by the user through their Graphic User Interface (GUI) applications,

b) after connection with MThings chosen by the user, the MThing notifies the user that the link has been established,

c) the user enters the role and execution order of each MThing. At this time, the user can describe or input the mission of each MThing through GUI, menu button interfaces, or natural language,

d) converts the user's input into the mission data using the corresponding standardised data format (e.g. XML),

e) the mission data (e.g. mission diagram) is presented to the user through the user's GUI to confirm it,

f) the mission data verified by the user is modified for each participating MThing and broadcast to all connected MThings,

g) the MThings start performing their respective tasks based on the transmitted mission data,

h) whenever each MThing completes its task recorded in the mission data, it broadcasts the completed status to other connected MThings (At this time, the user can check the task execution status of each MThing through their GUI),

i) and finally, when all tasks in the mission data are completed, the last MThing delivers status information notifying the completion of the mission to connected MThings. The user can check the notification about the mission completion through their GUI.

This process exemplifies one of the practical implementations to show the mission data creation and propagation, therefore, not restricted to it.

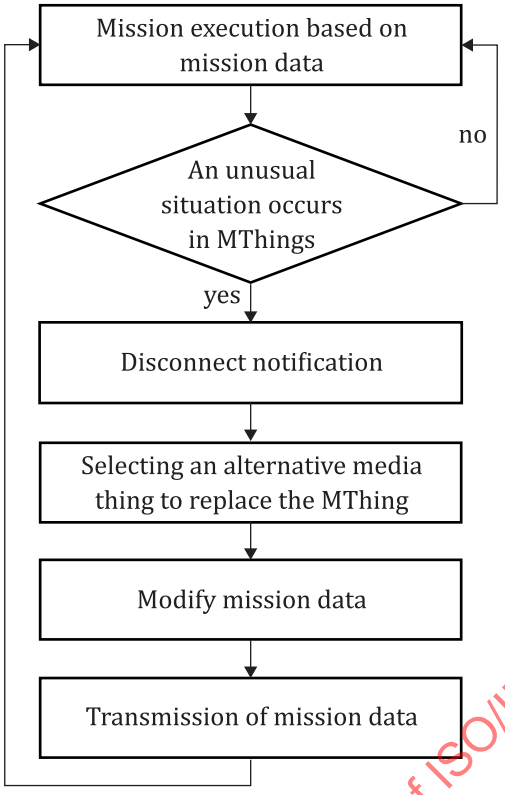

Figure 3 — Mission abort by an MThing

The following is an example of modifying the mission or stopping/excluding/modifying the related MThing due to a particular situation (e.g. malfunction or failure of an MThing) while performing the given mission.

— In case the connected MThing abandons a given task (e.g. `alertDisconnection()` defined in ISO/IEC 23093-2) (Figure 3)

  a) The MThing notifies all other connected MThings of its disconnection using the `alertDisconnection()` API defined in ISO/IEC 23093-2.

  b) The user checks this through the GUI and selects an MThing that can replace the disconnected MThing.

  c) The user modifies the task corresponding to the final state (replaced with a new MThing) using the state information, modifies and broadcasts the mission data, and performs steps D to I in Figure 2. At this time, MThings in the already completed state can release the connection or connected capability (e.g. using the `disconnectionMThing()` defined in ISO/IEC 23093-2). Mission data can also be modified accordingly.

In some cases, the above process can be replaced in a fully automated (more user-intervention) or autonomous (more AI-driven) manner.

— When the user arbitrarily aborts the mission by modifying the mission (Figure 4).

  a) When a user modifies a mission (e.g. adding a new MThing, adding a new function, or excluding an existing MThing), the currently executing mission can be forcibly aborted.

  b) The user modifies the mission data according to the purpose, broadcasts it to connected MThings, updates the mission, and performs steps C to I in Figure 2.

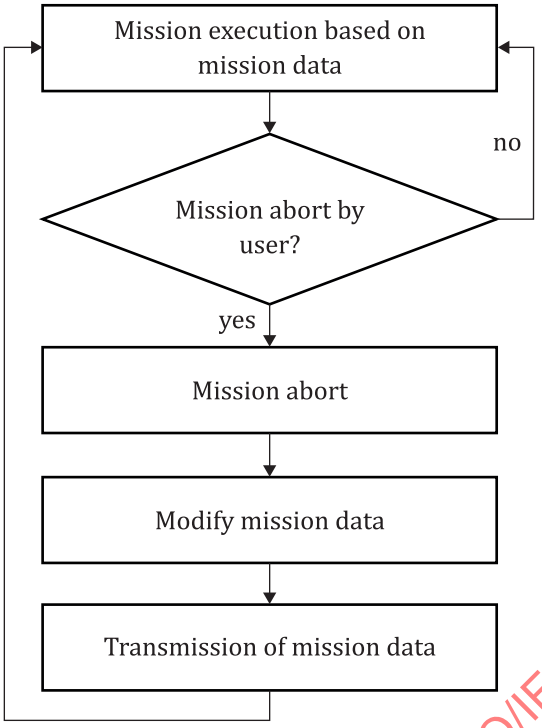

Figure 4 — Mission abort by a user

5.1.2.2 Mission process with MThing controller

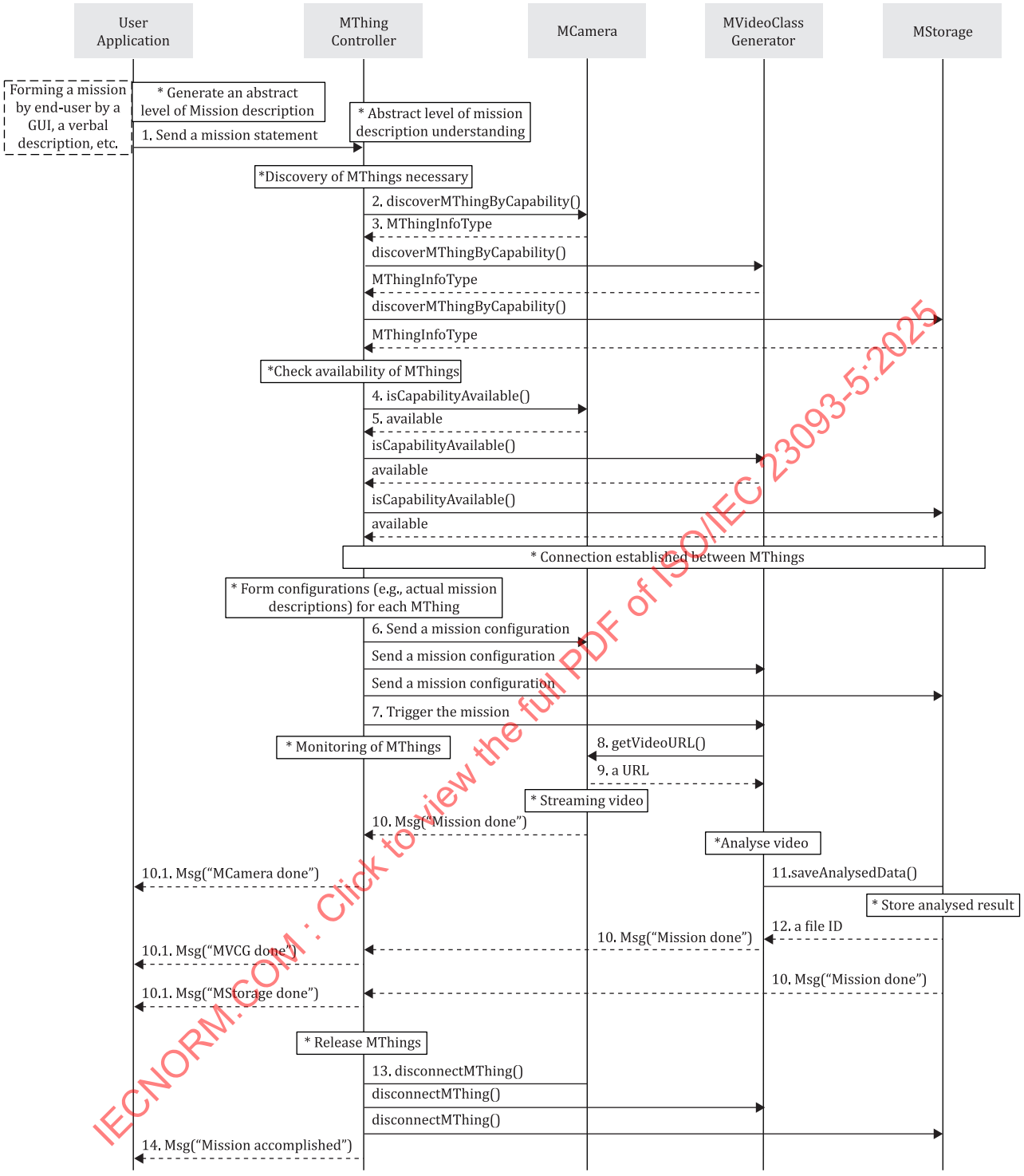

Figure 5 — Transition diagram with MController

The primary roles of the MThing controller (MController) IoMT autonomous operation are as follows:

a) Understand the abstract (e.g. holistic) level mission defined by the user through the application;

b) Search for MThings required for a mission, connect available MThings;

c) Create detailed (e.g. partial) level missions and deliver them to participating MThings;

d) Initiate, pause, resume, or abort the mission;

e) Monitor the status of participating MThings; and

f) Release the participating MThings after completing the mission.

Figure 5 shows a user and MThings sequence diagram, including the MController for autonomous operation. Suppose that a user wants to take a video from a nearby camera (i.e. MCamera), classify the captured video with an analyser (i.e. MVideoClassGenerator), and store the classified result in the storage (i.e. MStorage). The user can describe or input this task through a graph tool in the GUI, menu button interfaces, or natural language through the user applications.

The MController converts the task entered by the user into a holistic mission description (defined in subclause 6.2) that conforms to the standard. Here, the holistic mission description includes the types and functions of MThings required to complete the task and how the functions are executed. The holistic mission description generated by the user application is sent to the MController for understanding (Figure 5, item 1). The MController searches the network for MThings required to complete the mission based on the understood mission using the function `discoverMThingByCapability()` (defined in ISO/IEC 23093-2) (Figure 5, item 2). The discovered MThings return an `MThingInfoType` (defined in ISO/IEC 23093-3) containing their information to the MController (Figure 5, item 3). The MController asks whether the discovered MThing is available using the function `isCapabilityAvailable()` (defined in ISO/IEC 23093-2) (Figure 5, item 4) and notifies the MController that the MThings are available (Figure 5, item 5).

The MController and other MThings are connected using the connection API defined in ISO/IEC 23093-2. After completing the connection between MThings, the MController creates a detailed task description (e.g. the partial mission description defined in subclause 7.3) that each MThing must perform and sends the task description to all participating MThings. A partial mission description may include input and output information of each MThing, information on other MThings receiving output, and actions after task completion (Figure 5, item 6).

After the partial mission description is delivered to each MThing, the MController activates the mission of the MThing that starts the first mission (Figure 5, item 7). The MVideoClassGenerator checks its task and uses the function `getVideoURL()` (defined in ISO/IEC 23093-6) to request a URL for video streaming from MCamera (Figure 5, item 8). The MCamera returns the URL (Figure 5, item 9). Through the URL, the MVideoClassGenerator receives the streaming video as much as it requires. After streaming, the MCamera reports to the MController that its mission is complete (Figure 5, item 10). Upon receiving the mission completion message, the MController sends it back to the user application to notify them that the MCamera has completed the mission (Figure 5, item 10.1). Every message can be exchanged by the function `sendMessage()` (defined in ISO/IEC 23093-2) among MThings.

The MVideoClassGenerator, which has received video streaming from the MCamera, analyses the video and delivers the result to a pre-specified MStorage (Figure 5, item 11) using the function `saveAnalysedData()` (defined in ISO/IEC 23093-3). After saving the analysed result, the MStorage returns the file ID to the MVideoClassGenerator (Figure 5, item 12). The MStorage, which keeps the file, reports its mission completion to the MController, which reports it to the user. After confirming that all tasks have been completed, the MController releases all connected MThings using the function `disconnectMThing()` (defined in ISO/IEC 23093-2) (Figure 5, item 13) and delivers the final task completion message to the user (Figure 5, item 14).

5.1.2.3 Mission process without MController

The MController is expected to be helpful for the autonomous operation of large-scale IoMT systems or applications. However, in a small-scale IoMT system or application, e.g. IoMT Home, several MThings owned by the user should act as the MController. For example, through the user application of the newly purchased MCamera, the user can connect other MThings in the house and assign a task to complete the task automatically. The example below introduces the mission transition sequence to achieve the mission by giving the function of MController to the existing MCamera.

Figure 6 shows a user and MThings sequence diagram for autonomous operation. Suppose the same use case scenario is applied in Figure 5 to this example. The holistic mission description generated by the user application is sent to the MCamera for understanding (Figure 6, item 1). The MCamera searches the network for MThings required to complete the mission based on the understood mission using the function

`discoverMThingByCapability()` (defined in ISO/IEC 23093-2) (Figure 6, item 2). The discovered MThings return an MThingInfoType containing their information to the MCamera (Figure 6, item 3). The MCamera asks whether the discovered MThing is available using the function `isCapabilityAvailable()` (Figure 6, item 4) and notifies the MCamera that the MThings are available (Figure 6, item 5).

The MCamera and other MThings are connected using the connection API defined in ISO/IEC 23093-2. After completing the connection between MThings, the MCamera creates and sends a detailed task description (e.g. the partial mission description defined in subclause 7.3) that each MThing must perform and sends to all participating MThings (Figure 6, item 6). After the partial mission description is delivered to each MThing, the MCamera activates the mission of the MVideoClassGenerator that starts the first mission (Figure 6, item 7).

The MVideoClassGenerator checks its task and uses the function `getVideoURL()` to request a URL for video streaming from the MCamera (Figure 6, item 8), and the MCamera returns the URL (Figure 6, item 9). Through the URL, the MVideoClassGenerator receives the streaming video as much as it requires. After streaming, the MCamera reports to the user that its mission is complete (Figure 6, item 10) using the function `sendMessage()` (defined in ISO/IEC 23093-2).

The MVideoClassGenerator, which has received video streaming from MCamera, analyses the video and delivers the result to a pre-specified MStorage (Figure 6, item 11) using `saveAnalysedData()`. After saving the analysed result, the MStorage returns the file ID to the MVideoClassGenerator (Figure 6, item 12). The MStorage reports its mission completion to the MCamera, which reports it to the user (Figure 6, item 10.1). After confirming that all tasks have been completed, the MCamera releases all connected MThings using the function `disconnectMThing()` (Figure 6, item 13) and delivers the final task completion message to the user (Figure 6, item 14).

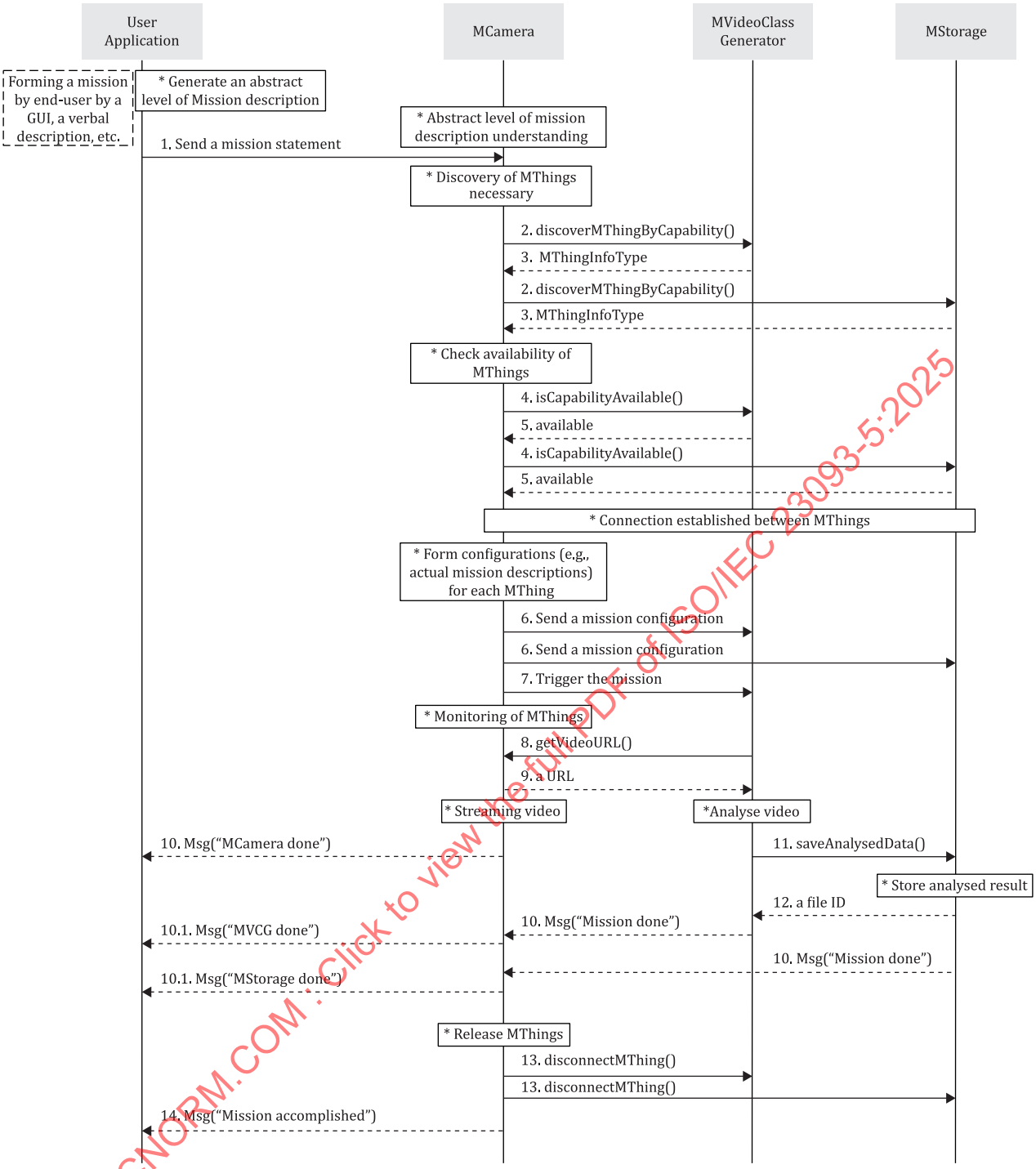

Figure 6 — Mission transition diagram without MController

5.1.2.4 Binding and unbinding process of MController

When an MController binds a group of MThings to accomplish the given mission, the MController should reserve the MThings until the completion of the assigned mission. In other words, the collaboration process described in Figures 5 and 6 has a structure in which the MThings participating in the collaboration can disconnect (through the function `disconnect()` defined in ISO/IEC 23093-2) only after their mission is completed.

In the case of large-scale collaboration, an MThing, which performs a task at the end, may face the problem of increasing its idle time. In addition, the MThing in the idle state cannot be used by other MControllers.

Figure 7 and the process description below depict the binding and unbinding process of the requesting MController. In step 5, the requesting MController can send its priority as a parameter when requesting unbind. The requested MController can determine the release of the requested MThing by checking the received priority. The MThings used by other collaborations can be either returned to their original mission or joined in another mission.

< MController requesting an MThing>

a) Find other nearby MControllers in charge of collaboration (`discoverMThingByCapability()`).

b) Receive a list of MThings included in the MController (`getMThingList()`).

c) Among MThings included in the list, search for MThings required. If the necessary MThing is found, proceed to step 4. If not, go back to step 1 to search for another MController.

d) Check whether the MThing required for the mission is currently performing its task (`isMThingBusy()`). If the MThing is busy, go back to step 3; otherwise, go to step 5.

e) Request Unbinding the corresponding MThing to another MController (`unbind(MThingInfo)`). If the corresponding MController succeeds in unbinding, the return value is 1; otherwise, the return value is 0. If the return value is 1, proceed to step 6; if it is 0, return to step 3.

f) Connect the MThing (`connectMThing()`). If the connection is successful, proceed to step 7. If not, go back to step 3.

g) Deliver the task description to the corresponding MThing.

h) Disconnect(release) the corresponding MThing when the mission is done (`disconnect()`).

i) Notify the original MController that the corresponding MThing has been released after completing the sub-task (`reconnectMThing(MThingInfo)`).

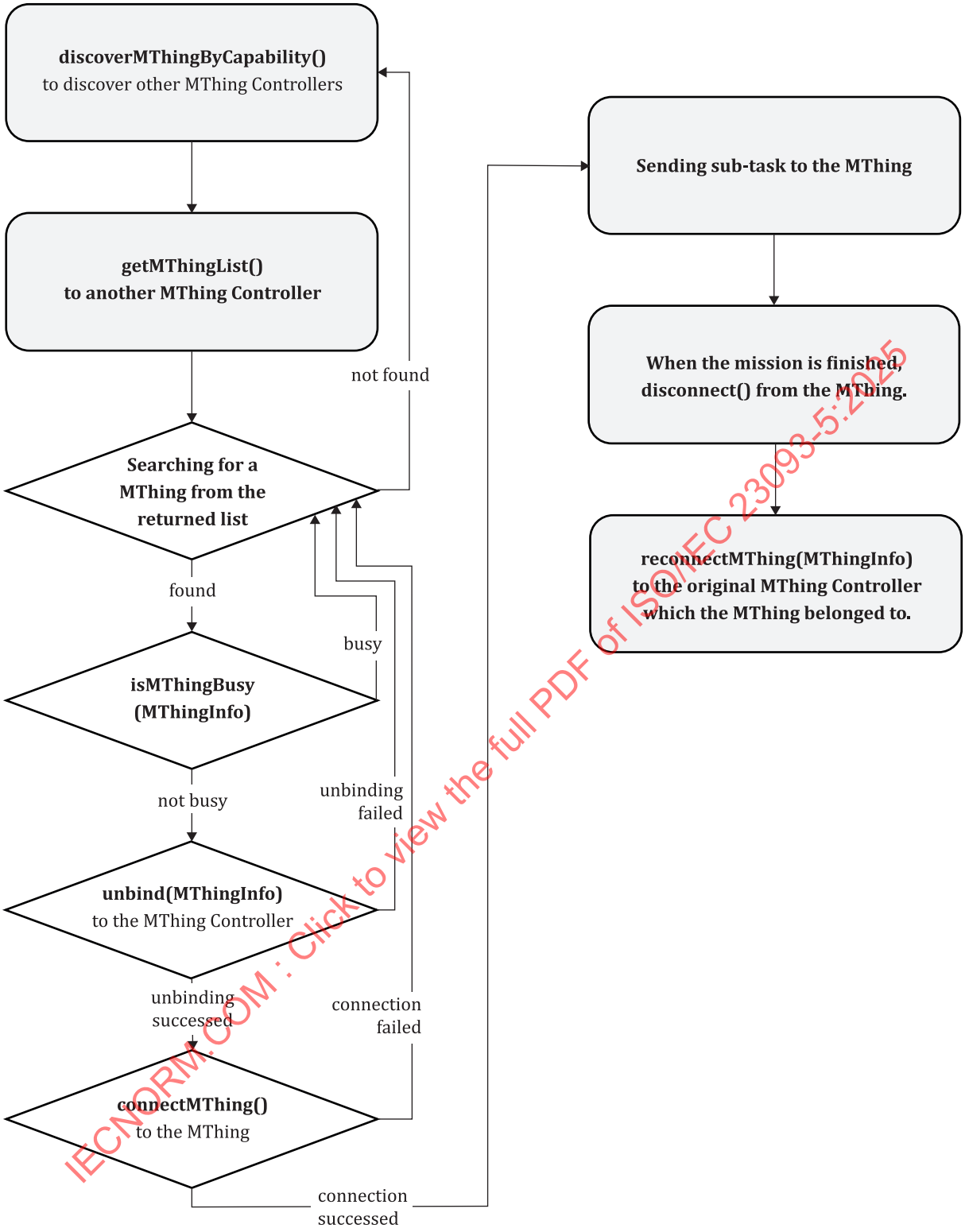

Figure 7 — Binding and unbinding process of the requesting MController

Figure 8 and the process description below depict the process of the requested MController. In step 3, the requested MController can determine the release of the MThing by priority, such as the estimated time it takes to use the requested MThing and the remaining steps in the mission sequence. Also, when releasing the requested MThing, an API may lend the MThing only to the requested MController and get it back only from the requested MController.

< MController yielding the requested MThing>

a) When `getMThingList()` is called, it returns the MThing list it contains for mission execution.

b) When receiving `isMThingBusy()`, it checks that the requested MThing performs its task and returns the result.

c) When `unbind(MThingInfo)` is called, the corresponding MThing is disconnected, and one is returned. If it is difficult to disconnect because the MThing is converted to the busy state or equivalent conditions, the 0 is returned.

d) When `reconnectMThing()` is called, the corresponding MThing is reconnected, the task of the corresponding MThing is assigned again, and the task execution continues. The notification is ignored if another MThing has already replaced the corresponding MThing.

e) Suppose the MThing of the same role is urgently needed before the corresponding MThing is returned. The MController starts searching for an MThing from another MController.

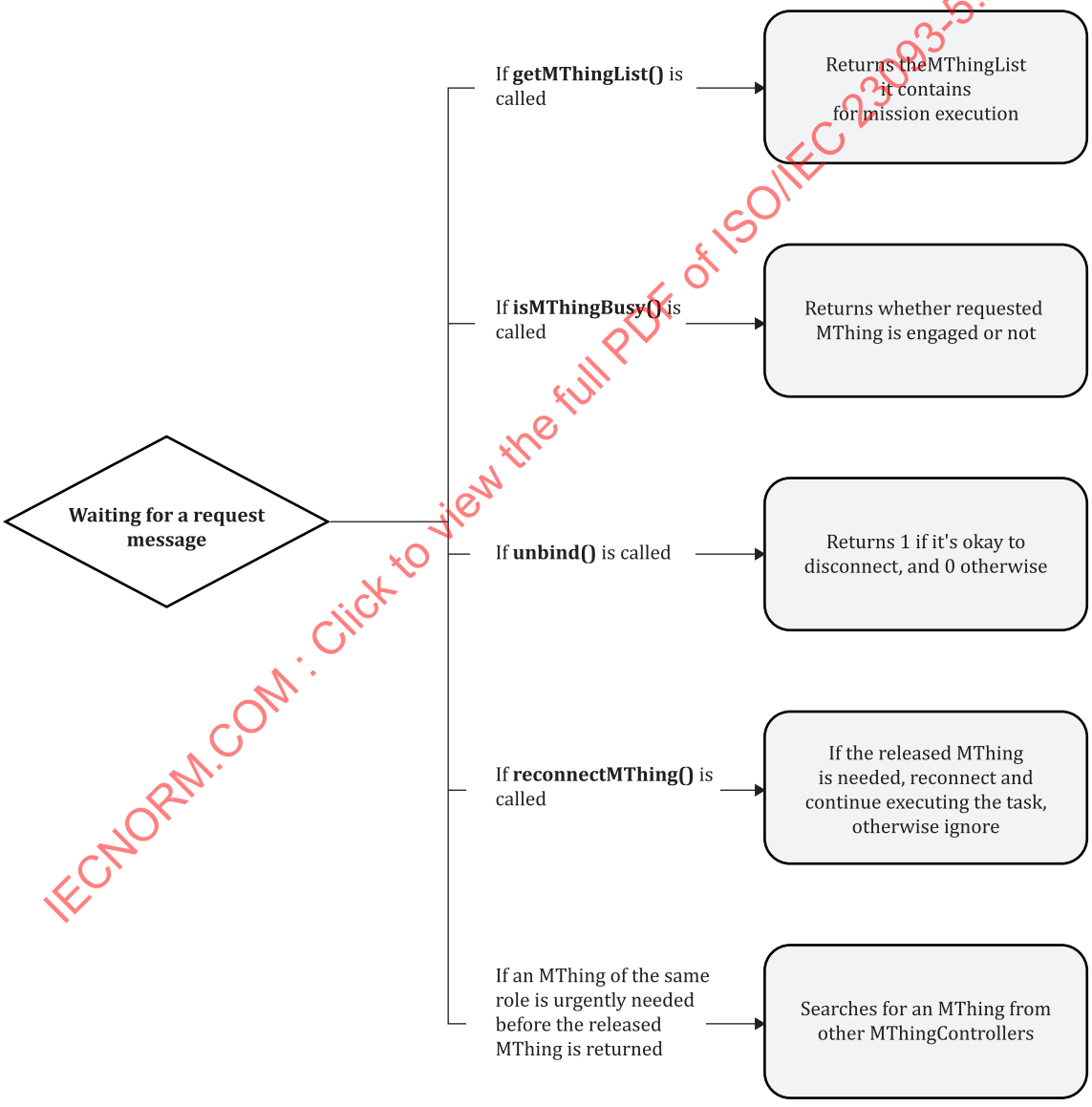

Figure 8 — Unbinding and binding process of the requested MController

5.1.2.5 Mission execution environment

The environment for performing the mission can be classified according to several criteria.

— According to the control method of mission performance:

  a) Centralised manner: a management server connects MThings participating in the mission and manages the overall mission performance, as discussed in subclause 5.1.2.2.

  b) Distributed manner: each MThing completes a task individually according to the mission data, as discussed in subclause 5.1.2.3.

  NOTE This document covers aspects of MController and mission description; in practice, the mission description can be used both with or without MController.

— According to how the mission data is delivered:

  a) Active transmission: a method in which a user or server broadcasts mission data.

  b) Passive transmission: a method in which MThings participate in the mission request and receive mission data stored in advance (e.g. using blockchain to keep the mission data).

— According to the number of MThings participating in the mission and its connection type:

  a) Small scale and ad-hoc connection: a small number of MThings are connected in an ad-hoc manner to perform the mission (e.g. connecting nearby objects to accomplish the assigned mission);

  b) Large scale and persistent connection: many MThings are connected to perform a mission collaboratively; those things are constant between successive tasks to be collaboratively performed.

For instance, suppose users can search for and connect MThings (e.g. displays, digital signage, surveillance cameras, speakers, and media analysers) to design and perform some basic service (i.e. mission) in the home environment. This type of scenario can be achieved in a distributed manner by active mission data transmission on a small scale and by ad-hoc connection between MThings.

A more stable and robust system is required when performing a service (i.e. mission) by connecting many MThings in a large setup (a hotel or office building). This time, the mission data stored in advance in the centralised system that manages each MThing reads the service passively to perform the task. A combination of a larger scale error-resilient and stable connection method can be configured.

6 APIs

6.1 General

This subclause defines the API classes of IoMT controllers.

6.2 MController class

6.2.1 General

This subclause defines an MController class that shall inherit the features of the `MThing` class defined in ISO/IEC 23093-2.

6.2.2 APIs

Table 1 presents the basic APIs of MController.

Table 1 — MContoller API

Nested Classes	
Modifier and Type	**Method and Description**
Constructor	
Constructor and Description	
MController()	
Default constructor.	
MController(string id)	
MController(string id, string serverIPAddress, int serverPort)	
Fields	
Modifier and Type	**Field and Description**
Methods	
Modifier and Type	**Method and Description**
CapabilityListType	getControllerCapabilityList()
	This function returns a class (i.e. Java or C++) or a structure (i.e. C) that shall include a returning type(e.g. XML, Binary) and a capability list specified in Annex A.
CapabilityListType	getAvailableControllerCapabilityList()
	This function returns a class (i.e. Java or C++) or a structure (i.e. C) that shall include a returning type (e.g. XML, Binary) and an available capability list specified in Annex A.
CapabilityListType	getAppliedControllerCapabilityList()
	This function returns a class (i.e. Java or C++) or a structure (i.e. C) that shall include a returning type (e.g. XML, Binary) and an applied capability list specified in Annex A.
List<MThingInfoType>	getMThingList()
	This function returns a list of MThings bound by the MController.
bool	isMThingBusy(MThingInfoType MThingInfo)
	This function checks whether or not the requested MThing is currently busy for the mission. The function returns 'true'; otherwise, it returns' false' if the MThing is busy.
int	unbind(MThingInfoType MThingInfo, int priority)
	If the corresponding MThing is disconnected, the function returns '1'. If it is difficult to disconnect because it is converted to a busy state or equivalent conditions, the function returns '0'. The `priority` indicates the importance level of the requesting side, where '0' is the highest priority, and the bigger number indicates the lower priority.
int	coupledUnbind(MThingInfoType MThingInfo, int priority)
	This function lends MThing only to the requesting MController. If the corresponding MThing is successfully disconnected and lent to the requesting MController, the function returns '1'. If it is difficult to disconnect because it is converted to a busy state or equivalent conditions, the function returns '0'. The `priority` indicates the importance level of the requesting side, where '0' is the highest priority, and the bigger number indicates the lower priority.
int	reconnectMThing(MThingInfoType MThingInfo)
	If the corresponding MThing is reconnected, the function returns '1'. If the corresponding MThing has trouble reconnecting MThing, this request is ignored, and the function returns '0'.
int	coupledReconnectMThing(MThingInfoType MThingInfo)

Table 1 *(continued)*

	This function receives MThing only from the MController that borrowed MThing and reconnects it. If the corresponding MThing is reconnected, the function returns '1'. If the corresponding MThing has trouble reconnecting MThing, the function returns '0'.
string	getMissionDescription(MThingInfoType mThingInfo)
	This function partial mission description.

7 Data formats

7.1 General

This subclause explains how and what to include in the mission description generated by the user interface (e.g. a holistic mission description) and by MController (e.g. a partial mission description).

7.2 Holistic mission data formats

7.2.1 General

This subclause describes the holistic mission description generated by the user interface application.

7.2.2 Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="https://www.w3.org/2001/XMLSchema" xmlns:mpeg7="urn:mpeg:mpeg7:sche
ma:2004">
    <xs:import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="https://standards.iso.
org/ittf/PubliclyAvailableStandards/MPEG-7_schema_files/mpeg7-v2.xsd"/>
    <xs:element name="MissionDescription">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="MissionObjective" type="xs:string" minOccurs="0"
maxOccurs="1"/>
                <xs:element name="ParticipatingMThingList" type="ParticipatingMThingListTy
pe"/>
                <xs:element name="TriggerMThing" type="AdditionalMThingInfoType"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Orders" type="OrderType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="ParticipatingMThingListType">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:element name="ParticipatingMThing" type="ParticipatingMThingType"
minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="ParticipatingMThingType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="MThingCapability">
                <xs:complexType>
                    <xs:attribute name="requiredCapability" type="xs:string" use="optional"/>
```

```
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attributeGroup ref="BasicMThingInfoBaseAttributes"/>
        <xs:attribute name="idRef" type="xs:string" use="optional"/>
    </xs:complexType>

    <xs:complexType name="AdditionalMThingInfoType">
        <xs:attributeGroup ref="BasicMThingInfoBaseAttributes"/>
        <xs:attribute name="idRef" type="xs:string" use="optional"/>
        <xs:attribute name="requiredCapability" type="mpeg7:termReferenceType"
use="required"/>
    </xs:complexType>

    <xs:complexType name="OrderType">
        <xs:sequence>
            <xs:element name="MThingFrom" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="MThingTo" minOccurs="0" maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:attributeGroup ref="BasicMThingInfoBaseAttributes"/>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>

    <xs:attributeGroup name="BasicMThingInfoBaseAttributes">
        <xs:attribute name="index" type="xs:decimal" use="optional"/>
        <xs:attribute name="mThing" type="mpeg7:termReferenceType" use="required"/>
    </xs:attributeGroup>
</xs:schema>
```


7.2.3 Semantics

Name	Definition
MissionDescription	It describes the mission's objectives, the MThings participating, and the order in which the mission is performed.
MissionObjective	A brief human-readable mission description.
ParticipatingMThingList	List of MThings participating in the mission.
TriggerMThing	It is the MThing that does the very first stage of the mission.
Orders	It describes the order in which MThings are performed to achieve a mission.
ParticipatingMThingListType	Tool for describing the list of MThings participating in the mission.
ParticipatingMThing	It describes the MThing participating in the mission.
ParticipatingMThingType	Tool for describing the MThing participating in the mission.
MThingCapability	It describes one or more capabilities that MThing can perform.
requiredCapability	It describes the capability of the corresponding MThing allocated to perform the mission.

Name	Definition
AdditionalMThingInfoType	Tool for describing MThing in more detail.
idRef	It is the unique identifier of MThing.
requiredCapability	It describes the capability of the corresponding MThing allocated to perform the mission.
OrderType	Tool for describing the execution order in a step.
MThingFrom	It is an MThing that initiates in a step.
MThingTo	It is an MThing that finishes in a step.
BasicMThingInfoBaseAttributes	Tool for describing MThing in more detail.
index	It is an index to differentiate between MThings of the same type when participating in a mission.
mThing	It describes the type of MThings.

7.2.4 Example

In this example, an MTimeSynchroniser, two MCameras, and an MStorage participate in the mission. The mission starts with the MTimeSynchroniser, which sends commands to the two MCameras, and the MCameras return the requested ones to the MTimeSynchroniser. The MTimeSynchroniser then commands MStorage. A detailed description of this example is shown in Annex B.1.

```
<MissionDescription>
   <MissionObjective>
      Synchronise the time of the video taken by the two cameras and save it to the storage.
   </MissionObjective>
   <ParticipatingMThingList>
      <participatingMThing mThing="MTimeSynchroniser">
         <MThingCapability requiredCapability="ANALYSER_SYNCHRONISE_TIME"/>
      </participatingMThing>
      <participatingMThing mThing="MCamera" index=1 idRef="CAM001">
         <MThingCapability requiredCapability="SENSOR_CAPTURE_IMAGE"/>
      </participatingMThing>
       <participatingMThing mThing="MCamera" index=2 idRef="CAM002">
         <MThingCapability requiredCapability="SENSOR_CAPTURE_IMAGE"/>
      </participatingMThing>
      <participatingMThing mThing="MStorage">
         <MThingCapability requiredCapability="STORAGE_SAVE"/>
      </participatingMThing>
   </ParticipatingMThingsList>

   <TriggerMThing mThing="MTimeSynchroniser" requiredCapability="ANALYSER_SYCHRONISE_TIME"/>

   <Orders>
      <MThingFrom mThing="MTimeSynchroniser">
         <MThingTo mThing="MCamera" index=1/>
        <MThingTo mThing="MCamera" index=2/>
      </MThingFrom>
      <MThingFrom mThing="MCamera" index=1/>
         <MThingTo mThing="MTimeSynchroniser"/>
      </MThingFrom>
      <MThingFrom mThing="MCamera" index=2/>
         <MThingTo mThing="MTimeSynchroniser"/>
      </MThingFrom>
```

```
      <MThingFrom mThing="MTimeSynchroniser"/>
         <MThingTo mThing="MStorage"/>
      <MThingFrom mThing=" MStorage"/>
         <MThingTo mThing="MTimeSynchroniser"/>
      </MThingFrom>
   </Orders>
</MissionDescription>
```

7.3 MController output vocabulary

7.3.1 General

This subclause specifies the syntax and semantics of the MController output vocabulary, which comprises the following media description:

— IoMT partial mission description.

NOTE MCOV has been designed as extensible, and additional media sensors can be added easily.

EXAMPLE Additional media sensors can be added as extensions to `mtdl:MissionDataBaseType` and conformance to MTDL.

7.3.2 Schema wrapper

The syntax of description tools specified in this subclause is a collection of schema components, including type definitions and element declarations. To form a valid schema document, users can gather these schema components in the same document with the following declaration defining, in particular, the target namespace and the namespaces prefixes.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
  xmlns=https://www.w3.org/2001/XMLSchema
  xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
  xmlns:mtdl="urn:mpeg:mpeg-IoMT:2024:01-MTDL-NS">

<import namespace="urn:mpeg:mpeg7:schema:2004" schemaLocation="https://standards.iso.org/
ittf/PubliclyAvailableStandards/MPEG-7_schema_files/mpeg7-v2.xsd"/>
<import namespace="urn:mpeg:mpeg-IoMT:2024:01-MTDL-NS" schemalocation=" https://standards.
iso.org/ittf/PubliclyAvailableStandards/MPEG-IOMT_schema_files/MPEG-IOMT-MTDL.xsd"/>
```

The following line should be appended to the resulting schema document to obtain a well-formed XML document.

```
</schema>
```

7.3.3 IoMT partial mission description

7.3.3.1 General

This subclause describes the partial mission description generated by the MController for each participating MThings.