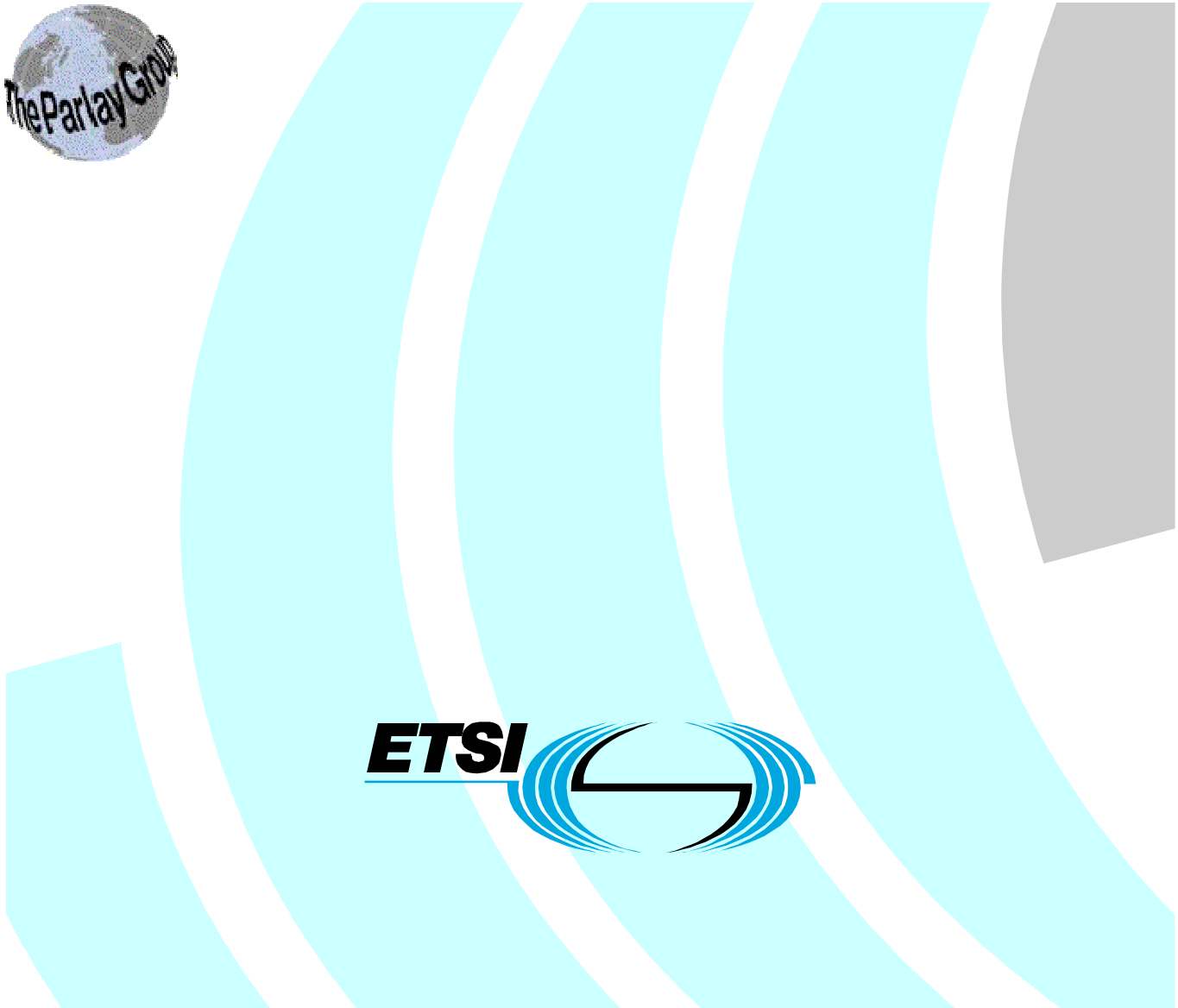
ETSI TR 102 397-5-1 V1.1.1 (2005-12)

Technical Report

Open Service Access (OSA);
Mapping of Parlay X Web Services to Parlay/OSA APIs;
Part 5: Multimedia Messaging Mapping;
Sub-part 1: Mapping to User Interaction



Reference
DTR/TISPAN-01021-05-01-OSA

Keywords
API, OSA, service

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN).

The present document is part 5, sub-part 1, of a multi-part deliverable providing an informative mapping of Parlay X Web Services to the Parlay Open Service Access (OSA) APIs and, where applicable, to IMS, as identified below:

Part 1: "Common Mapping";

Part 2: "Third Party Call Mapping";

Part 3: "Call Notification Mapping";

Part 4: "Short Messaging Mapping";

Part 5: "Multimedia Messaging Mapping";

Sub-part 1: "Mapping to User Interaction";

Sub-part 2: "Mapping to Multi-Media Messaging";

Part 6: "Payment Mapping";

Part 7: "Account Management Mapping";

Part 8: "Terminal Status Mapping";

Part 9: "Terminal Location Mapping";

Part 10: "Call Handling Mapping";

Part 11: "Audio Call Mapping";

Part 12: "Multimedia Conference Mapping";

Part 13: "Address list Management Mapping";

Part 14: "Presence Mapping".

The present document has been defined jointly between ETSI, The Parlay Group (http://www.parlay.org) and the 3GPP.

1 Scope

The Parlay X Web Services provide powerful yet simple, highly abstracted, imaginative, telecommunications functions that application developers and the IT community can both quickly comprehend and use to generate new, innovative applications.

The Open Service Access (OSA) specifications define an architecture that enables application developers to make use of network functionality through an open standardized interface, i.e. the Parlay/OSA APIs.

The present document is part 5, sub-part 1, of an informative mapping of Parlay X Web Services to Parlay/OSA APIs.

The present document specifies the mapping of the Parlay X Multimedia Messaging Web Service to the Parlay/OSA User Interaction Service Capability Feature (SCF).

2 References

For the purposes of this Technical Report (TR) the following references apply:

[1] ETSI TR 121 905: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Vocabulary for 3GPP Specifications (3GPP TR 21.905)".

[2] W3C Recommendation (2 May 2001): "XML Schema Part 2: Datatypes".

NOTE: Available at http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[3] ETSI TR 102 397-1: "Open Service Access (OSA); Mapping of Parlay X Web Services to Parlay/OSA APIs; Part 1: Common Mapping".

[4] W3C Note (11 December 2000): "SOAP Messages with Attachments".

NOTE: Available at http://www.w3.org/TR/SOAP-attachments

[5] ETSI TS 123 140: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Multimedia Messaging Service (MMS); Functional description; Stage 2 (3GPP TS 23.140)".

[6] IETF RFC 2822: "Internet Message Format".

NOTE: Available at http://www.ietf.org/rfc/rfc2822.txt

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 102 397-1 [3] and the following apply:

Shortcode: Short telephone number, usually 4 to 6 digits long. This is represented by the 'tel:' URI defined in TR 102 397-1 [3].

Whitespace: See definition for CFWS as defined in RFC 2822 [6].

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 102 397-1 [3] and the following apply:

MMS	Multimedia Messaging Service
SMS	Short Message Service

4 Mapping Description

The Multimedia Messaging capability can be implemented with the Parlay/OSA User Interaction SCF.

It is applicable to ETSI OSA 1.x/2.x/3.x, Parlay/OSA 3.x/4.x/5.x and 3GPP Releases 4.x/5.x/6.x.

5 Sequence Diagrams

5.1 Send Multimedia Message to One or More Addresses

This describes where an application sends a multimedia message to one or more addresses.

1. Prior to processing any **sendMessageRequest** messages from the application, the web service creates an event notification with criteria identifying the application (`OriginatingAddress`) and the terminal delivery related states (ServiceCode).

2. The application requests the sending of a multimedia message to multiple addresses using the **sendMessage** operation. If the contents of the **sendMessageRequest** message are invalid for any reason, the appropriate service or policy exception is thrown. Otherwise, a **sendMessageResponse** message is returned to the application containing a unique identifier for this multimedia delivery request and processing continues as described below.

3. The web service resolves all group addresses in the **addresses** part of the **sendMessageRequest** message to individual destination addresses. The web service creates a UI session for each individual destination address in the request.

4. The web service sends the message to each individual destination address and requests a message identifier (e.g. a network tracking number) using the `sendInfoAndCollectReq` method.

5. The application can invoke the **getMessageDeliveryStatus** operation at any time after it receives the **sendMessageResponse** message and use the unique identifier it received in this message to obtain the current delivery status for each individual destination address. At this stage, the status returned for each address is either **MessageWaiting** or, in the event of an error, **DeliveryImpossible**.

6. The web service processes an invocation of a `sendInfoAndCollectRes` method for each individual destination address, which contains a message identifier (e.g. a network tracking number).

7. After the web service processes the `sendInfoAndCollectRes` method invocation for a destination address, it releases the associated UI session objects created in step 3.

8. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for each individual destination address is one of the following:

   - **MessageWaiting**, if the `sendInfoAndCollectRes` method has not yet been invoked.

   - **DeliveryImpossible,** in the event an error occurred.

   - **DeliveryUncertain**, otherwise.

9. The web service processes an invocation of a `reportEventNotification` method containing the message identifier (i.e. as received by the web service in step 6), the terminal delivery related status and the sent message. This method notifies the application of an occurred network event matching specific terminal delivery related status criteria, which were previously installed with an invocation (in step 1) of the `createNotification` method.

10. *[RESERVED FOR FUTURE USE] If the **receiptRequest** part of the associated, original **sendMessageRequest** message was present, and this capability is supported by the web service, then the web service invokes the **notifyMessageDeliveryReceipt** operation to notify the application of the final status of the message delivery to an individual destination address.*

11. The web service releases the associated UI session object created in step 9.

12. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for each individual destination address is one of the following:

    - **Delivered**, if this status is reported by the `reportEventNotification` method.
    - **MessageWaiting**, if the `sendInfoAndCollectRes` method has not yet been invoked.
    - **DeliveryImpossible,** in the event an error occurred.
    - **DeliveryUncertain**, otherwise.

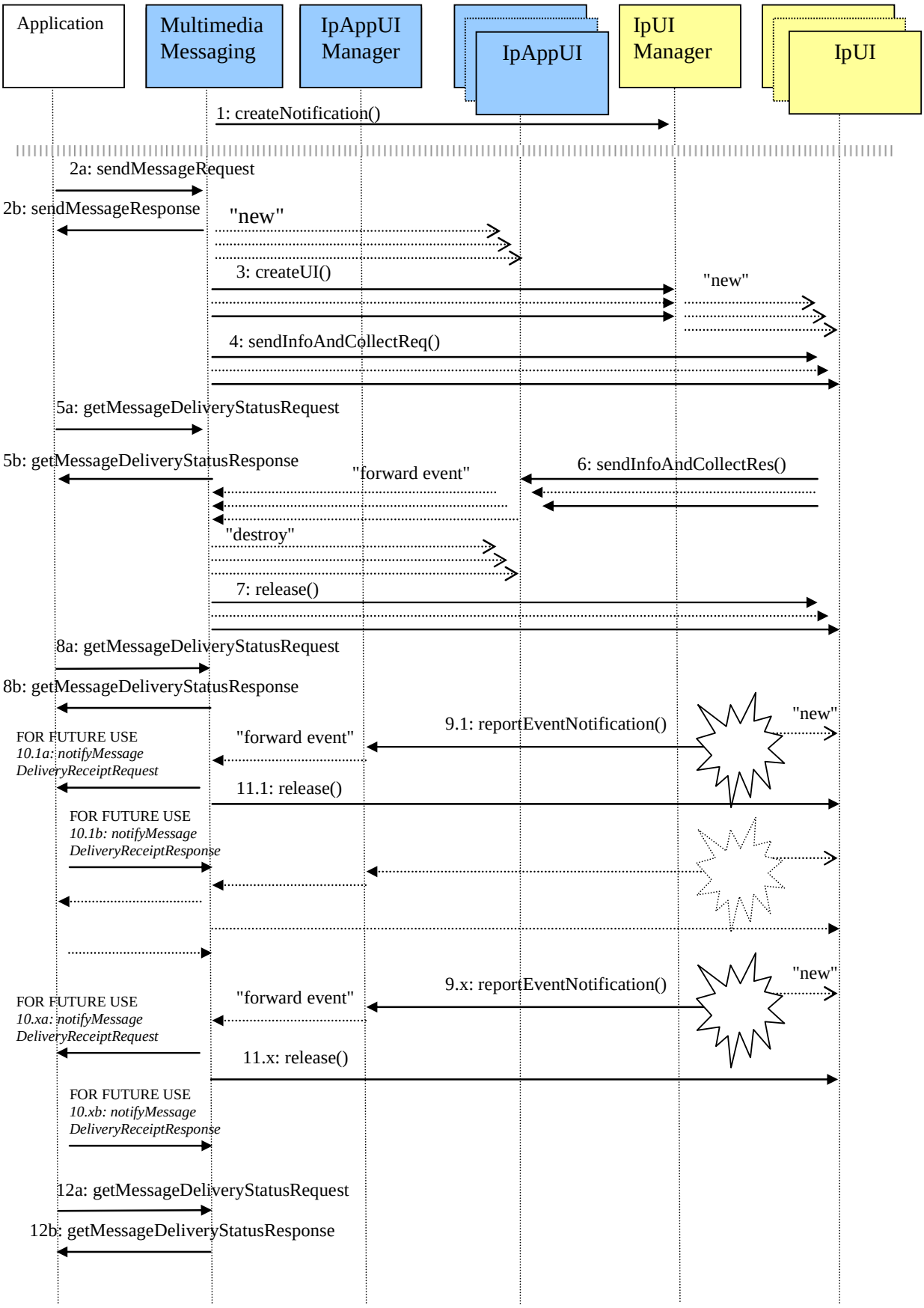

Application
Multimedia Messaging
IpAppUI Manager
IpAppUI
IpUI Manager
IpUI
1: createNotification()
2a: sendMessageRequest
2b: sendMessageResponse
"new"
3: createUI()
"new"
4: sendInfoAndCollectReq()
5a: getMessageDeliveryStatusRequest
5b: getMessageDeliveryStatusResponse
"forward event"
6: sendInfoAndCollectRes()
"destroy"
7: release()
8a: getMessageDeliveryStatusRequest
8b: getMessageDeliveryStatusResponse
9.1: reportEventNotification()
"new"
FOR FUTURE USE
10.1a: notifyMessage DeliveryReceiptRequest
"forward event"
11.1: release()
FOR FUTURE USE
10.1b: notifyMessage DeliveryReceiptResponse
9.x: reportEventNotification()
"new"
FOR FUTURE USE
10.xa: notifyMessage DeliveryReceiptRequest
"forward event"
11.x: release()
FOR FUTURE USE
10.xb: notifyMessage DeliveryReceiptResponse
12a: getMessageDeliveryStatusRequest
12b: getMessageDeliveryStatusResponse

Figure 1

5.2 Notification of Multimedia Message Reception and Retrieval

1. The application registers for the reception of multimedia messages by invoking **startMessageNotification**. The request includes event criteria consisting of a value for the multimedia message destination address (the **messageServiceActivationNumber** part) and an optional text string for matching against the first word of the subject of the multimedia message or the first word in the text part of the multimedia message (the **criteria** part); also a URI for a Web Service implementing the **MessageNotification** interface on the client application side, and a correlation value for identifying this event registration request.

   - Note that the application may also register offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **startMessageNotification** operation. The registration request should at a minimum specify the message destination address. The request may also specify a URI for a Web Service implementing the **MessageNotification** interface on the client application side and/or the optional text string criteria. The registration request is assigned a unique registration identifier.

2. A check is made within the web service to see if a notification for the given multimedia message destination address is active. If no notification is active, then the Multimedia Messaging web service requests that a notification be created by the UI SCS; note that the optional text string criteria (for matching against the first word in the message subject or body) is not sent to the UI SCS. Otherwise a notification is already active and the request is not made.

3. The UI SCS sends a `reportEventNotification` containing the message identifier, the message delivery status and the received message. The web service stores the multimedia message information.

4. The web service releases the UI session within the notification and verifies the event satisfies all criteria specified in step 1, including matching the first word of the message subject or body against the value of the optional text string criteria. If the event is verified, then it stores the received message and notifies the application (step 5); else the event is invalid (step 5 is skipped) and it discards the received message.

5. The web service notifies the application of the received multimedia message information by invoking the **notifyMessageReception** operation on the application Web Service. Note that if the multimedia message is pure ASCII text, then the whole message is delivered to the application Web Service.

6. Steps 3, 4 and 5 are repeated for any received message event matching the notification criteria. The application may invoke the **getReceivedMessages** operation to request a list of references to received multimedia messages matching a registration identifier associated with off-line provisioned notification criteria. The web service returns the list of any such multimedia message references. Note that for each multimedia message that is pure ASCII text, the web service delivers the whole message to the application and then deletes it.

7. The application retrieves the text portion of a multimedia message associated with one of the message references, and a list of URI file references for any message attachments, by invoking the **getMessageURIs** operation.

8. The application retrieves the whole multimedia message associated with one of the message references, by invoking the **getMessage** operation.

9. The application terminates an existing registration for the reception of multimedia messages by invoking the **stopMessageNotification** operation. The request includes the same correlation value previously specified in the earlier **startMessageNotification** operation (step 1).

   - Note that the application may also deregister offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **stopMessageNotification** operation. The deregistration request would specify the registration identifier associated with the original, offline registration operation (step 1).

10. A check is made within the web service to see if the registration identifier (correlation value) represents the last active notification for the corresponding destination address. If it is the last, then the web service requests that the notification be destroyed by the UI SCS. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the request is not made.

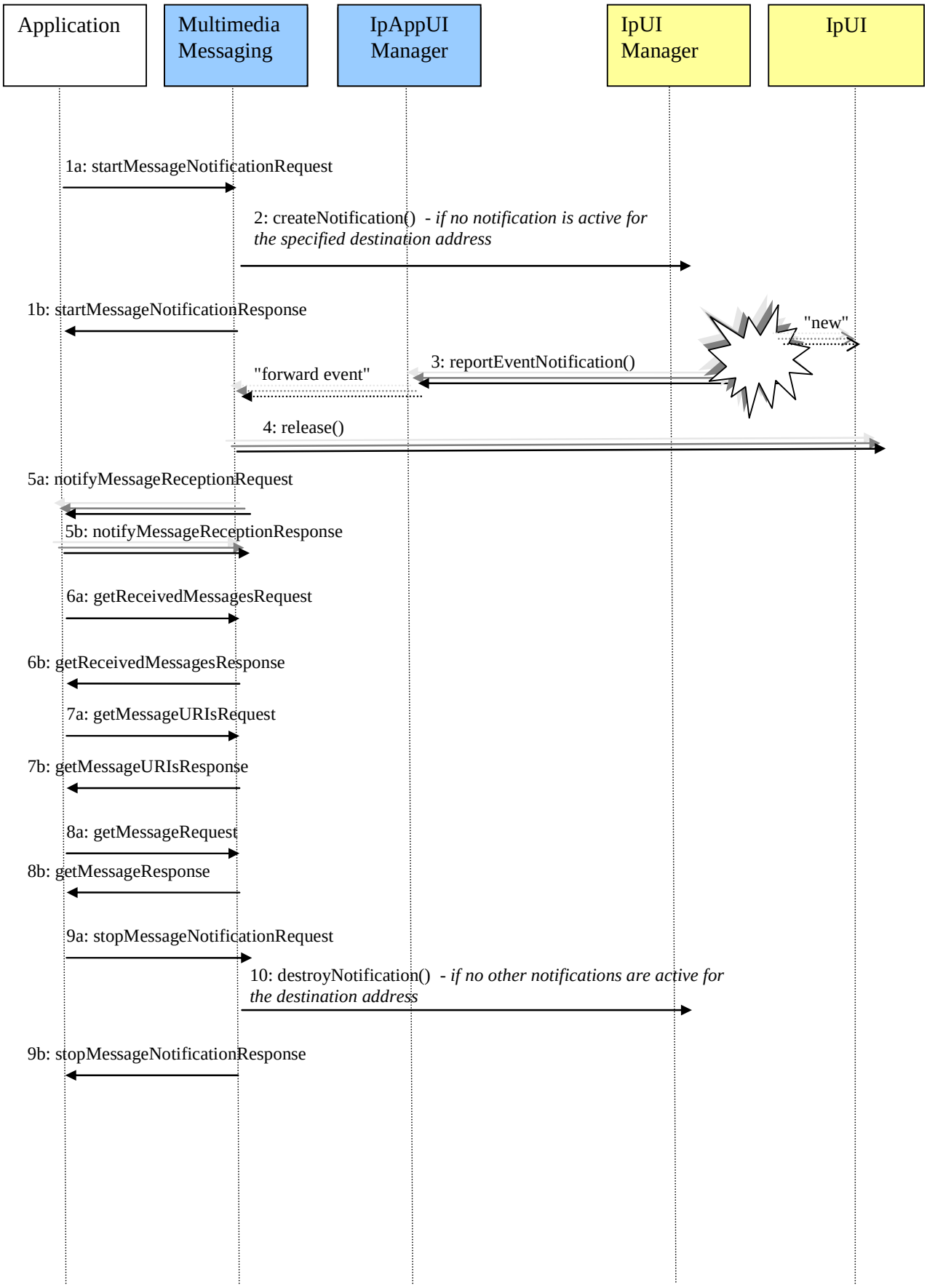

Application
Multimedia Messaging
IpAppUI Manager
IpUI Manager
IpUI
1a: startMessageNotificationRequest
2: createNotification() - if no notification is active for the specified destination address
1b: startMessageNotificationResponse
"new"
3: reportEventNotification()
"forward event"
4: release()
5a: notifyMessageReceptionRequest
5b: notifyMessageReceptionResponse
6a: getReceivedMessagesRequest
6b: getReceivedMessagesResponse
7a: getMessageURIsRequest
7b: getMessageURIsResponse
8a: getMessageRequest
8b: getMessageResponse
9a: stopMessageNotificationRequest
10: destroyNotification() - if no other notifications are active for the destination address
9b: stopMessageNotificationResponse

Figure 2

6 Detailed Mapping Information

6.1 Operations

6.1.1 sendMessage

The sequence diagram in clause 5.1 (figure 1) illustrates the flow for the **sendMessage** operation.

The **sendMessage** operation is synchronous from the Parlay X client's point of view. It is mapped to the following Parlay/OSA methods:

- `IpUIManager.createNotification;`
- `IpUIManager.createUI;`
- `IpUI.sendInfoAndCollectReq.`

6.1.1.1 Mapping to `IpUIManager.createNotification`

Prior to processing any **sendMessageRequest** messages from the application, the web service creates an event notification with criteria identifying the application (`OriginatingAddress`) and the terminal delivery related states (`ServiceCode`). The `IpUIManager.createNotification` method is invoked with the following parameters:

Name	Type	Comment
appUIManager	IpAppUIManagerRef	Reference to callback (internal).
eventCriteria	TpUIEventCriteria	The mapping to the `eventCriteria` parameter is as follows: • The `OriginatingAddress` element identifies the Parlay X application: e.g. as appropriate, the `Plan` element is assigned a value of `P_ADDRESS_PLAN_URL, P_ADDRESS_PLAN_SMTP` etc. • The `DestinationAddress` element is not mapped: i.e. the `Plan` element is assigned a value of `P_ADDRESS_PLAN_ANY`. • The `ServiceCode` element, which defines a 2-digit code indicating the UI to be triggered, is set to an operator-specific value identifying one or more terminal delivery related status event(s) to be monitored.

The result from `IpUIManager.createNotification` is of type `TpAssignmentID` and is used internally to correlate the callbacks. Specifically it is used to correlate a future invocation of the `IpAppUIManager.reportEventNotification` method, which reports a terminal delivery related status event for multimedia messages originated by this Parlay X application.

Parlay exceptions thrown by `IpUIManager.createNotification` indicate that the delivery receipt notification capability is not supported for this application. They are not mapped to Parlay X exceptions.

6.1.1.2 Mapping to `IpUIManager.createUI`

The `IpUIManager.createUI` method is invoked with the following parameters:

Name	Type	Comment
appUI	IpAppUIRef	Reference to callback (internal).
userAddress	TpAddress	Specifies the address to which the message should be sent. It is constructed based on the URI provided in the **addresses** part of **sendMessageRequest**, mapped as described in TR 102 397-1 [3].

The result from `IpUIManager.createUI` is of type `TpUIIdentifier` and identifies the User Interaction interface objects upon which future methods are invoked: e.g. `IpUI.sendInfoAndCollectReq`.

Parlay exceptions thrown by `IpUIManager.createUI` are not mapped to Parlay X exceptions. Instead they are reported to the application in a **getMessageDeliveryStatusResponse** message, with the following part values:

- the **deliveryStatus.address** element has the value of the address specified in the `userAddress` parameter of the `IpUIManager.createUI` method, mapped as described in TR 102 397-1 [3];
- the **deliveryStatus.deliveryStatus** element has the value: **DeliveryImpossible.**

6.1.1.3 Mapping to `IpUI.sendInfoAndCollectReq`

The `IpUI.sendInfoAndCollectReq` method is invoked with the following parameters:

Name	Type	Comment
userInteraction SessionID	TpSessionID	Not mapped. [The value provide in the result from `IpUIManager.createUI`].
info	TpUIInfo	There is no direct mapping for optional Attachments, W3C Note [4]. However there are several alternatives: • If the attachment(s) are pure text, then the content can be included in-band using the `InfoData` element. • If the attachment(s) are binary, then the content can be included in-band using the `InfoBinData` element or by using the `variableInfo` parameter (see below). • If the message is stored on a multimedia system, then its location (e.g. a URI) can be referenced using the `InfoData` or `InfoAddress` element; i.e. the message is sent out-of-band.
language	TpLanguage	Not mapped.
variableInfo	TpUIVariableInfo Set	• Some mapping support for the optional Attachments: the web service implementation can create local files for the attachments and provide the SCF with their URI references, by mapping them to `VariablePartAddress` element(s). • Some mapping support for the optional **messagePriority** part: i.e. it could be mapped to a `VariablePartInt` element. • Some mapping support for the optional **charging** part: i.e. it could be mapped to a `VariablePartPrice` element(s). • Some mapping support for the optional **senderAddress** part: i.e. it could be mapped to a `VariablePartAddress` element. • Some mapping support for the optional **subject** part: i.e. it could be mapped to a `VariablePartAddress` element. However, if this message is mapped to SMS, then this parameter will be used as the sender address, even if a separate **senderAddress** part is provided.
criteria	TpUICollect Criteria	Not mapped. Specifies additional properties for the collection of information from the network: i.e. a message identifier for the Multimedia Message.
response Requested	TpUIResponse Request	Not mapped. Set to `P_UI_RESPONSE_REQUIRED`.

The result from `IpUI.sendInfoAndCollectReq` is of type `TpAssignmentID` and is used internally to correlate the callbacks. Specifically it is used to correlate a future invocation of the `IpAppUI.sendInfoAndCollectRes` method.

Parlay exceptions thrown by `IpUI.sendInfoAndCollectReq` are not mapped to Parlay X exceptions. Instead they are reported to the application in a **getMessageDeliveryStatusResponse** message, with the following part values:

- the **deliveryStatus.address** element has the value of the address specified in the `userAddress` parameter of the `IpUIManager.createUI` method, mapped as described in TR 102 397-1 [3];
- the **deliveryStatus.deliveryStatus** element has the value: **DeliveryImpossible.**

6.1.2 getMessageDeliveryStatus

The sequence diagram in clause 5.1 (figure 1) illustrates the flow for the **getMessageDeliveryStatus** operation.

The **getMessageDeliveryStatus** operation is synchronous from the Parlay X client's point of view. It is mapped from the following Parlay/OSA methods:

- `IpAppUI.sendInfoAndCollectRes.`
- `IpAppUI.sendInfoAndCollectErr.`
- `IpAppUIManager.reportEventNotification.`

The delivery status provided to the Parlay X client will depend on the timing of the **getMessageDeliveryStatus** operation invocation. If a message event notification is triggered in the network as a result of an earlier **sendMessage** operation, then the delivery status information provided in the `IpAppUIManager.reportEventNotification` callback is mapped. If such a notification is not enabled, or it has not triggered, then the delivery status provided in the `IpAppUI.sendInfoAndCollectRes` callback is mapped.

6.1.2.1 Mapping from `IpAppUI.sendInfoAndCollectRes`

The `IpAppUI.sendInfoAndCollectRes` method is invoked with the following parameters:

Name	Type	Comment
userInteraction SessionID	TpSessionID	Not mapped. [The value provide in the result from `IpUIManager.createUI`].
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpUI.sendInfoAndCollectReq`].
response	TpUIReport	The `response` parameter maps to the **DeliveryUncertain** value of the **DeliveryStatus** element of a **DeliveryInformation** parameter of the **deliveryStatus** part of a **getMessageDeliveryStatusResponse** message.
collectedInfo	TpString	If the `response` parameter value is `P_UI_REPORT_INFO_COLLECTED`, then the `collectedInfo` parameter contains a network message identifier for the Multimedia Message. This identifier is subsequently used for correlating with the value of the `eventNotificationInfo.UIEventData` element of the `IpAppUIManager.reportEventNotification` method: clause 6.1.2.3.

6.1.2.2 Mapping from `IpAppUI.sendInfoAndCollectErr`

The `IpAppUI.sendInfoAndCollectErr` method is invoked with the following parameters:

Name	Type	Comment
userInteraction SessionID	TpSessionID	Not mapped. [The value provide in the result from `IpUIManager.createUI`].
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpUI.sendInfoAndCollectReq`].
error	TpUIError	Maps to the **DeliveryImpossible** value of the **deliveryStatus** element of a **DeliveryInformation** parameter of the **deliveryStatus** part of a **getMessageDeliveryStatusResponse** message.

6.1.2.3 Mapping from IpAppUIManager.reportEventNotification

The IpAppUIManager.reportEventNotification method is invoked with the following parameters:

Name	Type	Comment
userInteraction	TpUIIdentifier	Not mapped. Specifies the reference to the User Interaction interface and the sessionID to which the notification relates.

eventNotification Info	TpUIEvent NotificationInfo	The mapping to the **deliveryStatus** part is as follows:
		• The OriginatingAddress element is not mapped. It identifies the Parlay X application, as described in clause 6.1.1.1
		• The DestinationAddress element maps to the **DeliveryInformation.address** element.
		• The ServiceCode element contains an operator-specific value reporting a terminal delivery related status event. It is (one of) the value(s) specified in the ServiceCode element of the eventCriteria parameter of the IpUIManager.createNotification method (clause 6.1.1.1). This operator-specific value maps to one of the following values of the **DeliveryInformation.deliveryStatus** element:
		  • **DeliveryImpossible.**
		  • **Delivered.**
		• The DataTypeIndication element is not mapped, but should have a value of P_UI_EVENT_DATA_TYPE_TEXT.
		• The UIEventData element (a text string) provides the correlation with the UI interface objects used to send the message to the destination address. [It contains the message identifier returned to the web service in the collectedInfo parameter of the IpAppUI.sendInfoAndCollectRes method (clause 6.1.2.1).]

assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from IpUIManager.createNotification].

The result from IpAppUIManager.reportEventNotification is of type IpAppUIRef and is used internally to correlate with the User Interaction interface instance (i.e. of type IpUI) associated with the event notification. This callback reference result parameter may be set to a default value since there is no further interaction with this message delivery status-related UI instance: the IpUI.release method is invoked as shown in clause 5.1 (step 11).

6.1.3 startMessageNotification

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **startMessageNotification** operation, which is mapped to the Parlay/OSA method: IpUIManager.createNotification, provided there is no existing notification already established for the destination address contained in the **messageServiceActivationNumber** part.

6.1.3.1 Mapping to IpUIManager.createNotification

The IpUIManager.createNotification is invoked with the following parameters:

Name	Type	Comment
appUIManager	IpAppUIManagerRef	Not mapped. Reference to callback (internal).
eventCriteria	TpUIEvent Criteria	Specifies the event notification criteria, consisting of 3 elements: • The OriginatingAddress is not mapped. It is set to be valid for all senders. • The DestinationAddress is constructed based on the URI provided in the **messageServiceActivationNumber** part of the **startMessageNotificationRequest** message, mapped as described in TR 102 397-1 [3]. • The ServiceCode element is not mapped.

The result from IpUIManager.createNotification is of type TpAssignmentID and is used internally to correlate the callbacks. Specifically it is correlated with the value of the **reference** part received from the application in the **startMessageNotificationRequest** message and the **correlator** part returned to the application in the **notifyMessageReceptionRequest** message.

Note that the **reference** part and the optional **criteria** part of a **startMessageNotificationRequest** message are not mapped to `IpUIManager.createNotification`. Instead the web service uses all the text string criteria values associated with a specific destination address to parse any event reported for that address by the `IpAppUIManager.reportEventNotification` method. The web service determines whether the event is valid - i.e. there is a match with a text string criteria value. If valid, the web service stores the message and selects the previously provisioned application callback web service to receive the **notifyMessageReceptionRequest** message. If invalid, the web service discards the event notification.

Parlay exceptions thrown by `IpUIManager.createNotification` are mapped to Parlay X exceptions as defined in clause 6.2.

6.1.4 notifyMessageReception

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **notifyMessageReception** operation, which is mapped from the Parlay/OSA method: `IpAppUIManager.reportEventNotification`.

6.1.4.1 Mapping from `IpAppUIManager.reportEventNotification`

The `IpAppUIManager.reportEventNotification` method is invoked with the following parameters:

Name	Type	Comment
userInteraction	TpUIIdentifier	Not mapped. Specifies the reference to the User Interaction interface and the sessionID to which the notification relates.
eventNotification Info	TpUIEvent NotificationInfo	The mapping to the **message** part is as follows: • The `OriginatingAddress` element maps to the **senderAddress** element. • The `DestinationAddress` element maps to the **messageServiceActivationNumber** element. • The `ServiceCode` element is not mapped. • If the event-related message is ASCII text, then the `DataTypeIndication` element has a value of `P_UI_EVENT_DATA_TYPE_TEXT`, and the `UIEventData` element should contain the message, using a vendor/operator-specific convention, which maps to the **message** element. In this case the **messageIdentifier** element is absent. • If the event-related message is not ASCII text, then the `UIEventData` element should contain the message, using a vendor/operator-specific convention – also see note below. The multimedia message is stored by the Parlay X Multimedia Messaging Web Service. The latter returns a reference to this stored message in the **messageIdentifier** element. In this case the **message** element is absent.
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpUIManager.createNotification`].

Note that there is no direct mapping for Attachments. Binary content may be included in-line in the `UIEventData` element of the `eventNotificationInfo` parameter. Alternatively, the messaging system implementation could create local file(s) for the attachment(s) and provide the SCF with their URI reference(s). These URI reference parameters and others – e.g. that map to **fileReferences**, **priority**, **bodyText** and **subject** parts - could also be encoded in the `UIEventData` element of the `eventNotificationInfo` parameter.

Note that this mapping occurs if there is at least one active notification established for the value of the `eventNotificationInfo.DestinationAddress` element, an associated application callback web service, and one of the following conditions is satisfied:

- There is only one active notification that was defined without the optional text string criteria value.
- There is one active notification that was defined with the optional text string criteria value and that value matches the first word in the value of the `eventNotificationInfo.UIEventData` element.
  - Note that the 'first word' in the message is defined as the initial characters after discarding any leading Whitespace and ending with a Whitespace or end of message. The matching shall be case-insensitive.

The result from `IpAppUIManager.reportEventNotification` is of type IpAppUIRef and is used internally to correlate with the User Interaction interface instance (i.e. of type `IpUI`) associated with the event notification.

6.1.5 getReceivedMessages

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getReceivedMessages** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getReceivedMessages** operation is a bulk retrieval capability for previously received multimedia messages matching criteria defined in an off-line provisioning step. This retrieval operation includes matching messages previously and individually reported to the application via the **notifyMessageReception** operation.

6.1.6 getMessageURIs

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getMessageURIs** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessageURIs** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.1.7 getMessage

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getMessage** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessage** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.1.8 stopMessageNotification

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **stopMessageNotification** operation, which is mapped to the Parlay/OSA method: `IpUIManager.destroyNotification`, provided that the referenced notification is the last active notification for the associated destination address. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the mapping is not performed.

6.1.8.1 Mapping to IpUIManager.destroyNotification

The `IpUIManager.destroyNotification` is invoked with the following parameters:

Name	Type	Comment
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpUIManager.createNotification` and correlated with the value of the **reference** part received from the application in the original **startMessageNotificationRequest** message and the value of the **correlator** part received from the application in the **stopMessageNotificationRequest** message].

Parlay exceptions thrown by `IpUIManager.destroyNotification` are mapped to Parlay X exceptions as defined in clause 6.2.

6.2 Exceptions

For this mapping document, the mapping of Parlay/OSA API method exceptions to Parlay X Web Service exceptions is common and defined in TR 102 397-1 [3]. There are no service-specific exception mappings.

7 Additional Notes

No additional notes are provided.

History

Document history		
V1.1.1	December 2005	Publication