Deliverable D3.3

LPM Communication Standard

DISSEMINATION LEVEL		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	**X**
CO	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	**INTUITEL**
Project Full Name:	Intelligent Tutoring Interface for Technology Enhanced Learning
Grant Agreement No.:	318496
Programme	FP7-ICT-2011.8, Challenge 8.1
Instrument:	STREP
Start date of project:	01.10.2012
Duration:	33 months
Deliverable No.:	D3.3
Document name:	LPM Communication Standard
Work Package	3
Associated Task	3 and 4
Nature [1]	R
Dissemination Level [2]	RE
Version:	1.0
Actual Submission Date:	2013-09-30
Contractual Submission Date	2013-09-30
Editor: Institution: E-mail:	Oscar Garcia Perales, Luis de la Fuente Valentín TIE, UNIR oscar.garciaperales@intuitel.eu, luis.delafuente@intuitel.eu

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2011.8, Challenge 8.1) under grant agreement n°318496.

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

Change Control

Document History

Version	Date	Change History	Author(s)	Organization
0.1	2013-08-29	Document drafted	Oscar Garcia Perales	TIE
0.2	2013-09-03	Ch. 3 drafted General comments	Luis de la Fuente Valentín	UNIR
0.3	2013-09-06	Completely rewriting of Ch. 1 and 2	Oscar Garcia Perales José Rafael Navalón Baixeras	TIE
0.4	2013-09-10	Inclusion of Ch. 3	Luis de la Fuente Valentín	UNIR
0.5	2013-09-11	Comments solving	Oscar Garcia Perales	TIE
0.6	2013-09-16	Comments provided Ch. 3 finished	Florian Herbele, Peter Henning Luis de la Fuente Valentín	HSKA UNIR
0.7	2013-09-17	Comments solving	Oscar Garcia Perales	TIE
0.99	2013-09-24	Final version	Oscar Garcia Perales	TIE
1.0	2013-09-30	Approval	Peter Henning	HSKA

Distribution List

Date	Issue	Group
2013-09-02	Request for contribution	UNIR
2013-09-10	Request for comments	WP03
2013-09-17	Request for comments	WP03

Table of Contents

List of figures

List of tables

List of Abbreviations

Term	Explanation
API	Application Programming Interface
B2B	Business to Business
EDI	Electronic Data Interchange
FIFO	First In First Out
GPL	General Public License
HTTP	HyperText Transfer Protocol
JAAS	Java Authentication and Authorization Service
JMS	Java Messaging Service
JSON	JavaScript Object Notation
LMS	Learning Management System
LO	Learning Object
LORE	Learning Object Recommender
REST	Representational State Transfer
RPC	Remote Procedure Call
SSL	Secure Socket Layer
SOAP	Simple Object Access Protocol
TSB	TIE SmartBridge
TUG	Tutorial Guidance
UI	User Interface
USE	User Score Extraction
UTF	UCS Transformation Format
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

1 Communication Layer Design

1.1 Introduction

The Communication Layer (henceforth denoted CL) is a component of the INTUITEL system, which is responsible for the exchange of messages and data between components including the different Learning Management Systems (LMS). It provides a messaging service that can be used for any kind of remote service call. Two versions of the CL will be implemented:

- A "Basic" CL. The basic CL is a central component with basic routing and queuing features. These basic routing and queuing features have been developed from scratch as the basic CL is intended for being used in smaller, local and secure INTUITEL settings.
- An "Advanced" CL. The advanced CL is, like the basic one, a central component but as its name denotes it offers advanced routing and queuing features. In the advanced CL, these features will be provided by a middleware, which also provides with additional functionality, like advanced transformations or workflow editions. In addition to the usage of the middleware, an Application Server will be used for being the entry point of messages to the CL. This configuration is intended for larger, distributed INTUITEL settings in heterogeneous networks.

The type of CL shall not affect the functionality, the development and the configuration of the other INTUITEL components; therefore, it will be possible to switch between the two CL types.

1.2 Generic Architecture

1.2.1 Component Structure

The INTUITEL Communication Layer is compound of the following subcomponents as can be seen in Figure 1:

- Receiver: The receiver subcomponent is the entry point for the CL when it receives a message.
- Dispatcher: Similarly, the dispatcher subcomponent is the exit point for the CL when it sends a message.
- Message Processor: Finally, the message processor is the main subcomponent of the CL. The aim of the message processor is to process the messages, route and buffer them for final dispatch.

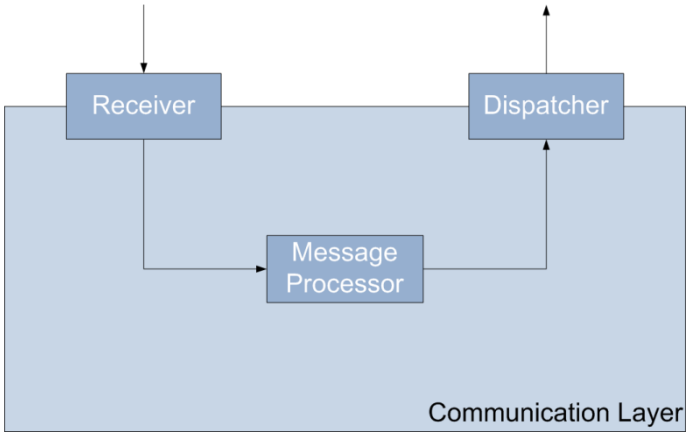

Figure 1: Architectural view of the Communication Layer

1.2.2 Location of the Communication Layer

As can be seen in Figure 2, the CL is located in the middle of every communication within the INTUITEL system (including the different LMSs). The three blocks shown in the top of the figure and the block shown in the bottom of the figure refer to the following systems:

- INTUITEL LPM End-point: This communication interface encapsulates all transmission related aspects and connects the Back End to the other INTUITEL components concerning the LPM elements, such as the LPM or the INTUITEL Engine.
- INTUITEL SLOM End-point: The SLOM communication interface enables the SLOM editor to exchange data with the LMS and the other INTUITEL components to access the SLOM Metadata Repository.
- INTUITEL Engine End-point: This communication interface encapsulates all transmission related aspects and connects the Back End to the other INTUITEL components concerning the Engine elements, such as the Reasoning Engine or Broker, the Natural Language Unit or the Recommendation Rewriter.
- LMS End-point: This communication interface encapsulates all transmission related aspects in the LMS. This enables the LMS Integrator to decouple the INTUITEL functionality with the technical and protocol issues of the data exchange. User and LO data as well as recommendations are sent through this end-point.

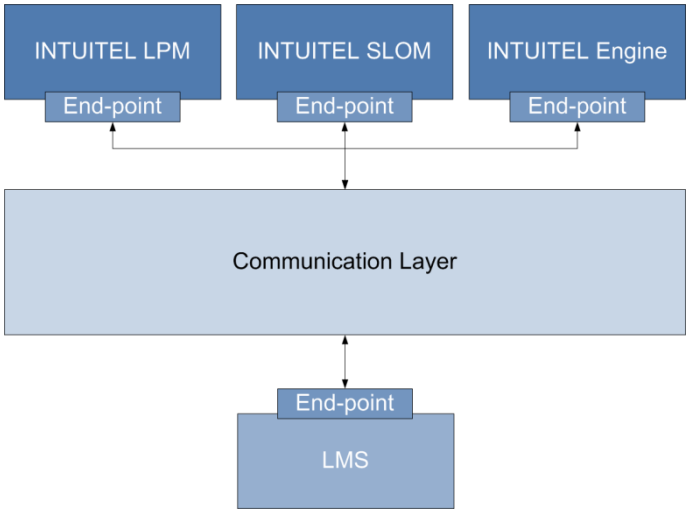

Figure 2: Block diagram showing the location of the Communication Layer

1.2.3 Sequence diagram

This section shows the sequence diagram that the CL will execute regardless the version of the CL selected. As such, the following steps are performed (shown in Figure 3):

- Step 1: The Communication Initiator sends a message. This Communication Initiator can be whichever component in the INTUITEL system, including the LMSs.
- Step 2: The Receiver subcomponent of the CL receives the message and sends it for internal processing to the Message Processor.
- Step 3: The Message Processor opens the message and process it for analysing for errors and searching of the URL of the destination.
- Step 4: The Message Processor sends the message to the Dispatcher for final dispatching.
- Step 5: The Dispatcher gets the message from the Message Processor and sends it to the Communication Recipient. Similarly to Step 1, the Communication Recipient can be whichever other component in the INTUITEL system, including the LMSs.

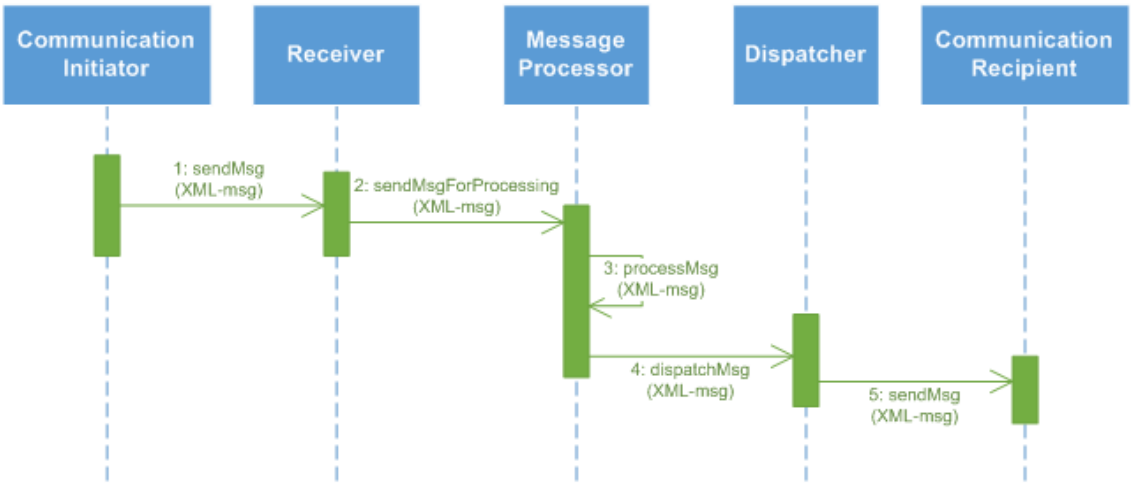

Figure 3: Sequence diagram of the Communication Layer

1.3 INTUITEL "Basic" Communication Layer

1.3.1 Introduction

As said in the introduction, INTUITEL will provide with two versions of the Communication Layer, the basic and the advanced ones. This section describes the basic version of the CL.

1.3.2 Component Structure

The INTUITEL "Basic" CL is based on the Communication Layer presented in the previous section. The only change is the Message Processor and the rest of the structure remains unchanged. The Message Processor is broken down in the following subcomponents, as can be seen Figure 4:

- Message Analyzer: The Message Analyzer is the first subcomponent of the Message Processor. Its activity relies on two different tasks:
    - Firstly, the Message Analyzer opens the XML-based message received and performs a syntactic analyse of the message received. If the XML is well formed then, the message is sent to the Routing Table subcomponent. Else, an XML message informing about the bad formulation is sent as response to the origin component of the message.
    - Secondly, the Message Analyzer seeks in the Properties Recognition file (see description in Table 1) a message type according to the content of the XML-based message.
- Routing Table: The Routing Table is the second subcomponent of the Message Processor. Its main objective is to store all the possible destinations for each message. Once a message has been processed and its properties have been recognized, the Routing Table is asked for a list of available routes that match the message type coming from the Message Analyzer. The Routing Table selects the most specific route (URL) of the possible ones. Once this URL is returned, the message is sent to the FIFO Queue for final dispatch.
- FIFO Queue: The FIFO queue is in charge of buffering the messages to be dispatched. When a message reaches the top of the queue, it is sent to the final destination.

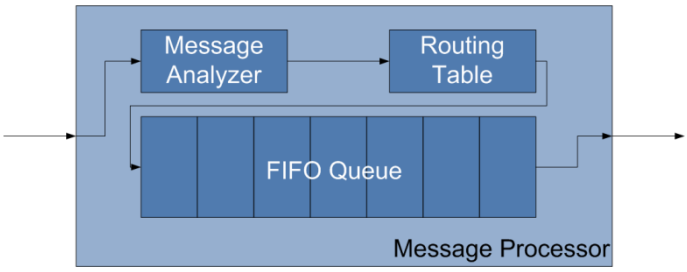

Figure 4: Basic Message Processor ("Basic" Communication Layer)

Both the Message Analyzer and the Routing Table make use of the following configuration files to, first recognize and second route an incoming message to the appropriate component recipient. These files are compound of the parameters shown in Table 1:

File	Parameter	Description
Components Configuration	Component ID	This field indicates the ID (including the URL) of any INTUITEL component. These components might be also the LMSs
	Services Provided	This field specifies the list of the different services provided by the component
Properties Recognition	Message Type	This field indicates the name of the messages configured to be recognized
	Recognition Property	This field indicates which are the recognition properties assigned to a given message type. These properties can be XPath, DTD, XSD or XML namespace. If more than one value is provided, the property set is the addition of all the values
Routes Configuration	Message Type	As before, this field indicates the name of the messages configured to be processed by the Routing Table
	Required Service	This field indicates the method to be executed when the Routing Table processes a message of that type. This field might include a list of additional parameters that might override the parameters initially specified by the method
Log	Component ID	This field indicates which component sent the message identified by the other parameter (Message ID)
	Message ID	This field indicates the ID of the message sent by the component identified by the parameter which type of message is configured

Table 1: List of Configuration Files needed in the INTUITEL "Basic" Communication Layer

1.3.3 Sequence diagram of the "Basic" Communication Layer

This section shows the sequence diagram for the "Basic" CL will execute. As such, the following steps are performed (shown in Figure 5):

- Step 1: The Communication Initiator sends a message. This Communication Initiator can be whichever component in the INTUITEL system, including the LMSs.
- Step 2: The Receiver subcomponent of the CL receives the message and sends it for internal processing to the Message Analyzer.
- Step 3: The Message Analyzer opens the message and analyse it for errors, like XML bad formed or if it is not conformant to the XML Schema. If the XML message is well formed, then the message is sent for the next step, else an error message is created and sent back to the Communication Initiator.

- Step 4: The Message Analyzer sends the message to the Routing Table subcomponent.
- Step 5: The Routing Table subcomponent opens the XML message and looks in its internal routing configuration file (according to Table 1) for the URL of the Communication Recipient.
- Step 6: The Routing Table subcomponent sends the message to the FIFO Queue.
- Step 7: Once the message has reached the top of the queue, this is sent to the Dispatcher for final dispatching.
- Step 8: The Dispatcher gets the message from the FIFO Queue and sends it to the Communication Recipient. Similarly to Step 1, the Communication Recipient can be whichever other component in the INTUITEL system, including the LMSs.

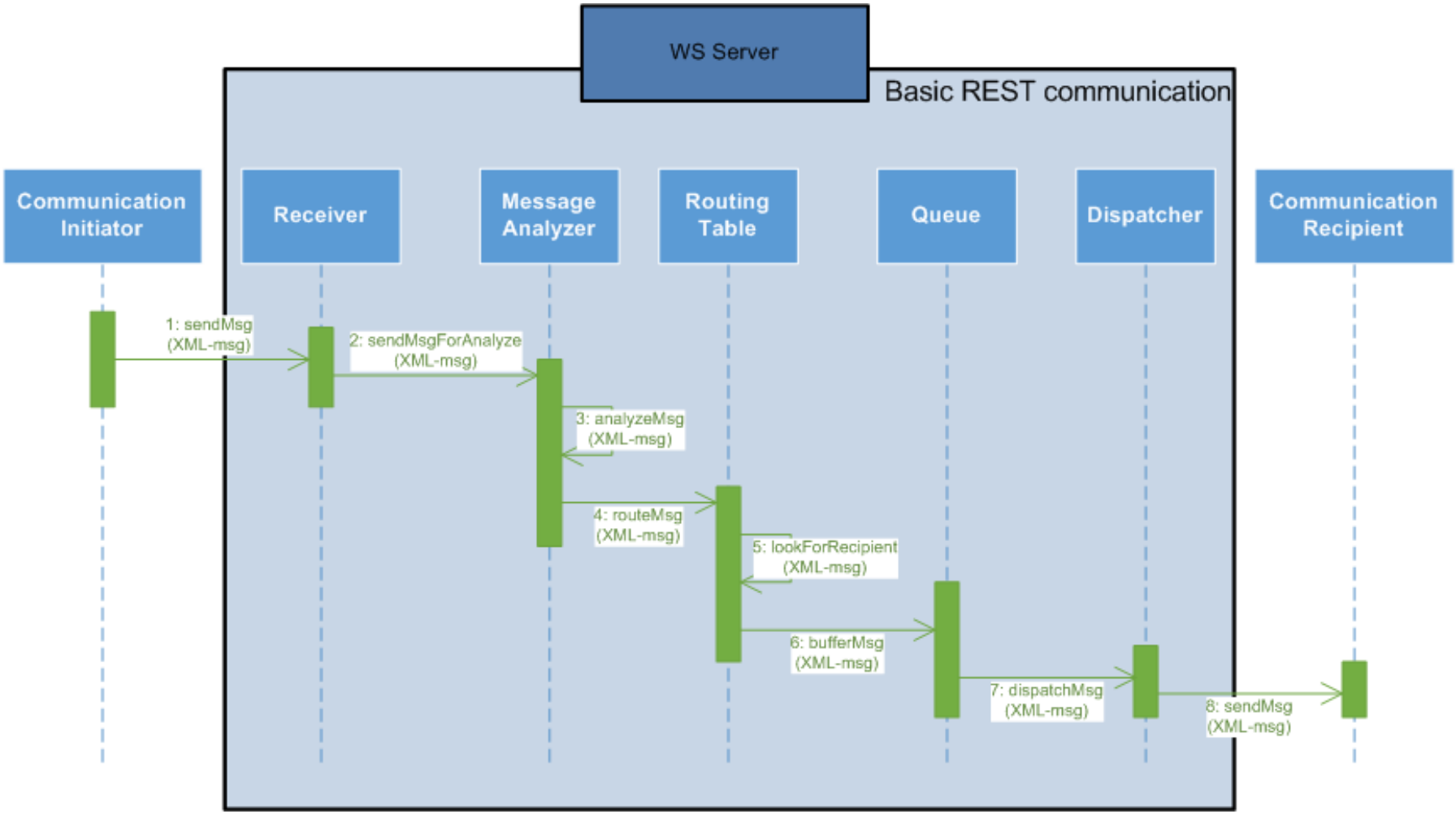

Figure 5: Sequence diagram of the "Basic" Communication Layer

1.4 INTUITEL "Advanced" Communication Layer

1.4.1 Introduction

As said in the introduction, INTUITEL will provide with two versions of the Communication Layer, the basic and the advanced ones. This section describes the advanced version of the CL.

1.4.2 Component Structure

The INTUITEL "Advanced" CL is based on the basic Communication Layer presented before. In addition to the "Basic" features, this version adds more features. The main change relates to the Message Processor although the Recipient and Dispatcher components are developed as part of an XMPP server. Thus, the Advanced CL is based on a two-layer architecture: a combination of TIE SmartBridge henceforth denoted as TSB, and an XMPP server.

On one hand, TSB is an advanced document analysing and message routing based system in which the connected components can send messages for some specific method, and services listening to those methods can then respond to those incoming messages. TSB is asynchronous in nature. The TSB supports the following core functions without extra implementation:

- Managing the valid interfaces.
- Facilitating the communication between the components.
- Messaging queues.
- Transforming between internal document structure and external document structure.
- Workflow management for complex operations.

On the other hand, the entry point to the "Advanced" CL will be done through an XMPP-based server. This technology incorporates other advanced functionality like:

- Connection with the TSB as processing middleware and external components by upholding the socket connection.
- Presence control, i.e. the delivery of an XMPP message can be delayed until the recipient is connected.
- Distributed systems, as the entry point is publicly available to all external components.

The "Advanced" CL therefore will be realized as a hybrid solution XMPP-TSB. The main advantage it has relies on the implementation side: each INTUITEL component will use the same calls as before, and the internals are completely hidden from the components through the usage of TSB. With this solution, some INTUITEL components, like e.g. the LMSs, will interact with the XMPP server (which is in charge of routing the messages to the TSB), while the INTUITEL backend will implement the TSB as it will access the services offered by INTUITEL, like e.g. the Engine. The structure of the "Advanced" CL can be seen in the following Figure 6:

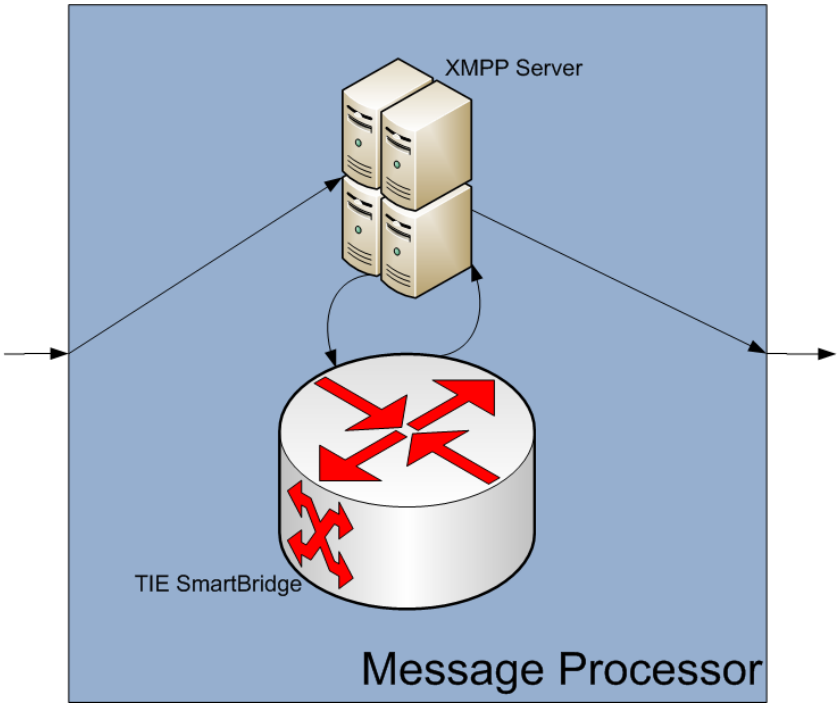

Figure 6: Advanced Message Processor ("Advanced" Communication Layer)

The internal structure of the TSB is compound of, among others, the following elements:

- Document Archiver: This subcomponent is used for logging and traceability purposes. Every change in the status of a message (or document) is recorded in this internal database. With this extra functionality, the system will easily recover from unexpected exceptions.
- Document Analyzer: Similarly to the Message Analyzer, this subcomponent opens the message and analyses the content of the XML sent checking for inconsistencies or bad formatting.
- Document Processing: This subcomponent is in charge of executing internal transformations. This extra functionality is useful if, e.g. the messages exchanged are following the same structure but expressed in different languages (as an example, <urlAddress> vs. <direccionUrl>) or there are differences in the XML tags (as an example, <urlAddress> vs. <url>).
- Routing: This subcomponent is in charge of looking the internal database of registered components and providing with the URL and credentials (if necessary) of the different end-points.
- Queue: This subcomponent is in charge of buffering the messages prior to their dispatching.

The XMPP will take over the responsibility of buffering the messages as the recipient of the communication might be unavailable and, thus, the message has to be transmitted anyway. It is responsibility of the recipient component of disregard or process buffered messages. In any case, the Advanced CL will just send the buffered messages. The XMPP protocol defines a strict policy for

clients to post a status that shows them as available when they are online to circumvent the buffering of messages. Messages can be buffered client-side this way. However, this does not work for INTUITEL, where messages need to be buffered and sent as soon as the recipient becomes available.

1.4.3 Sequence diagram of the "Advanced" Communication Layer

This section shows the sequence diagram that the "Advanced" CL will execute. As such, the following steps are performed (shown in Figure 7):

- Step 1: The Communication Initiator sends a message. This Communication Initiator can be whichever component in the INTUITEL system, including the LMSs.
- Step 2: The Receiver subcomponent of the CL (in the XMPP server) receives the message and sends it to the TSB which, in two sub-steps:
    - Step 2.1: sends the message to the Document Archiver for traceability purposes. The message is marked as "received".
    - Step 2.2: sends the message for internal processing to the Document Analyzer.
- Step 3: The Document Analyzer opens the message and analyse it for errors, like XML bad formed. If the XML message is well formed, then the message is sent for the next step, else an error message is created and sent back to the Communication Initiator.
- Step 4: The Document Analyzer sends the message to:
    - Step 4.1: the Document Archiver. The message is marked as "analyzed".
    - Step 4.2: the Document Processing.
- Step 5: The Document Processing opens the message and, if necessary, it performs any kind of transformations, like e.g. a XSLT Transformation.
- Step 6: Once transformed (if necessary), the Document Processing sends the message to:
    - Step 6.1: the Document Archiver. The message is marked as "transformed".
    - Step 6.2: the Routing.
- Step 7: The Routing subcomponent opens the XML message and looks in its internal database for the URL of the Communication Recipient.
- Step 8: The Routing subcomponent sends the message to:
    - Step 8.1: the Document Archiver. The message is marked as "buffered".
    - Step 8.2: the Queue.
- Step 9: Once the message has reached the top of the queue, this is sent to:
    - Step 9.1: the Document Archiver (still in the TSB). The message is marked as "sentForDispatching".
    - Step 9.2: the Dispatcher (in the XMPP server) for final dispatching.
- Step 10: The Dispatcher gets the message from the Queue and sends it to:
    - Step 10.1: the Document Archiver (back to the TSB). The message is marked as "dispatched".
    - Step 10.2: the Communication Recipient. Similarly to Step 1, the Communication Recipient can be whichever other component in the INTUITEL system, including the LMSs.

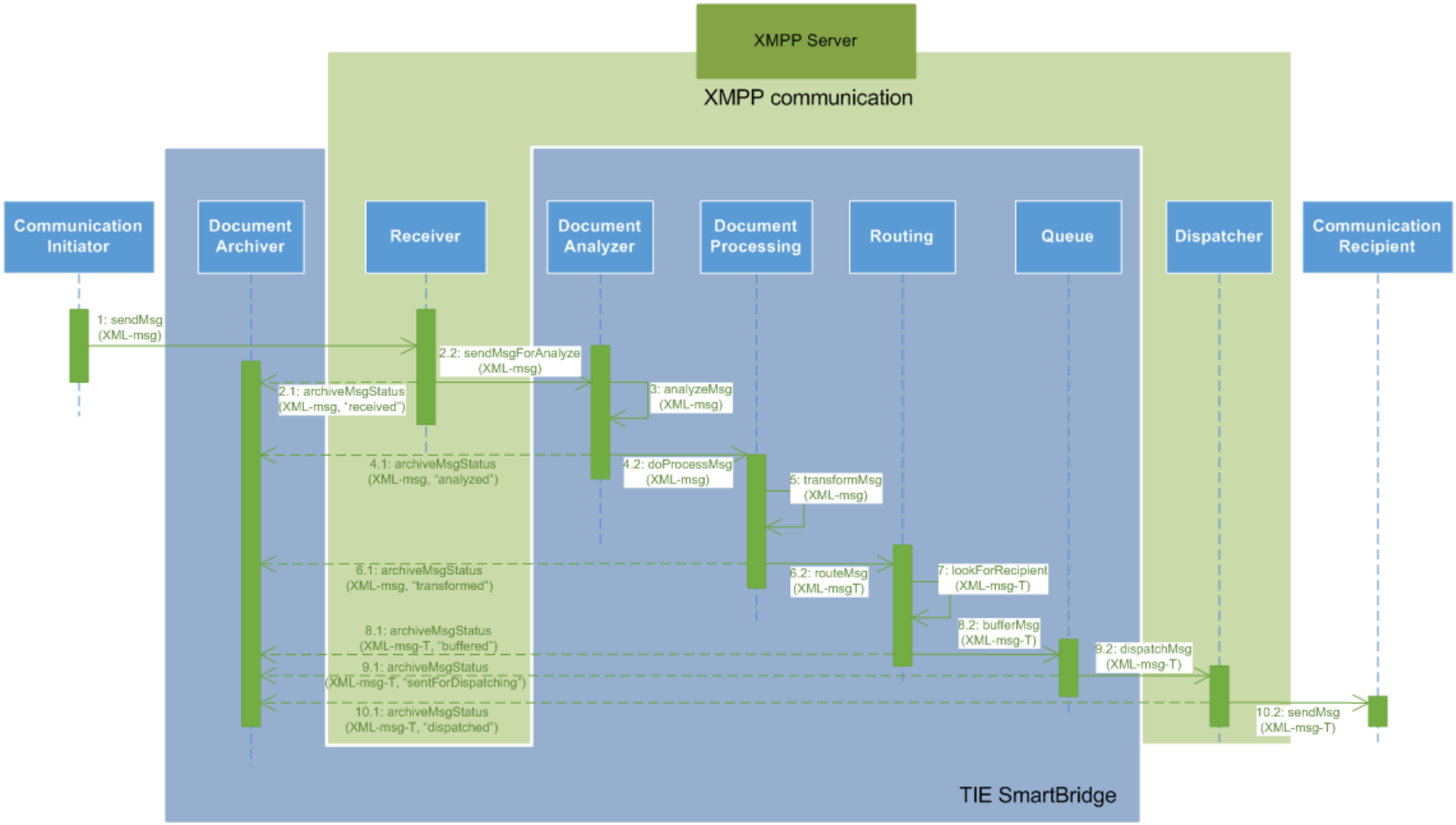

Figure 7: Sequence diagram of the "Advanced" Communication Layer

2 Communication Layer Development

2.1 Common API: Specification of Interfaces and Formats

To communicate with other components in the INTUITEL architecture all components use the Communication Layer API described in the following subsections. The formats of the messages exchanged have already been described in D1.1.

Furthermore, this API specification will be followed in whichever scenario for INTUITEL is used, either the "Basic" or the "Advanced" connectivity. It describes the different methods accessible when the different components in the INTUITEL Architecture, meaning internal INTUITEL components like the LPM or the INTUITEL Engine or non-INTUITEL components like e.g. the LMSs, need to invoke for communicating with each other.

As already said, in the "basic" scenario, the communication between the components is implemented as a REST web service on both sides through the "Basic" CL. The communication is achieved by the sender when passing the completed XML document to a specialized library (e.g. as a "stream") and this library will enclose it in an HTTP request sent to a web service in the secure environment. The HTTP request is sent through the "Basic" CL to the recipient and, in turn, this is then answered by an HTTP response delivered back to the sender. REST is a series of design guidelines for web services[3], hence not a strict way to express communication paradigms.

As outlined before, a simple FIFO message buffer (called Communication Manager in the Data Model D1.1 deliverable) will ensure that such calls are non-blocking and will increase message throughput performance (see Figure 4).

In the "advanced" scenario, the communication will be carried out through the "Advanced" CL. In other words, the caller will send an XMPP message over HTTP protocol to the XMPP server. Then, the XMPP server will send the request to the TSB, which will process the request. The message will reach its destination and the HTTP response will be sent back following the same procedure. Although the original and "native" transport protocol for XMPP is TCP (by using open-ended XML streams over long-lived TCP connections), the XMPP community has also developed an HTTP transport for web clients as well as users behind restricted firewalls. According to the XMPP specification, two possibilities of using XMPP are allowed: polling and binding. Although *polling* alternative has been recently deprecated, it allowed an XMPP client access messages stored on a server-side database by means of HTTP GET and POST requests. The *binding* alternative implies that servers push messages to clients as soon as they are sent by means of *Bidirectional-streams Over Synchronous HTTP* (BOSH). The reason that the *poll* alternative has been deprecated from the XMPP specification is that the latter alternative, *bind*, is more efficient as many of the polls returned no new data. Because the client uses HTTP, most firewalls allow clients to fetch and post messages

[3] See http://rest.elkstein.org for a concise description of this technology

without any hindrances. Thus, in scenarios where the TCP port used by XMPP is blocked, a server can listen on the normal HTTP port and the traffic should pass without problems. A perhaps more efficient transport for real-time messaging is WebSocket, a web technology providing for bi-directional, full-duplex communications channels over a single TCP connection.

In the "Advanced CL", the message has to be formatted following XMPP format, e.g. adding XMPP features not supported by REST; however, the body of the information remains the same.

The following table exhibits a list of (pseudo-)URLs that will be used in the LMS when implementing this web service, the method of transferring parameters as well as the HTTP response codes. This table continues Table 1 from the Data Model D1.1 deliverable.

URL	Description
<INTUITELEndPoint>/lmsprofile	Transferring the information related to the LMS profile
<INTUITELEndPoint>/learners	Transferring the information related to the progress of the learners
<INTUITELEndPoint>/mapping	Transferring the information related to the mappings of the LOs stored in the SLOM repository
<INTUITELEndPoint>/login	Transferring the information related to the credentials for Content Creators
<INTUITELEndPoint>/TUG	Transferring TUG messages
<INTUITELEndPoint>/LORE	Transferring LORE messages
<INTUITELEndPoint>/USE/performance	Transferring USE messages
<INTUITELEndPoint>/USE/environment	

Table 2: INTUITEL URL and high-level description of their role

2.1.1 External API description

The following sections describe the classes that will form the CL and the set of asynchronous Web Services belonging to each class. In addition, many of the current development frameworks do not need any specification of the call-back methods as they can be easily developed from the original calling Web Service.

2.1.1.1 Class LMSProfile

```
Class LMSProfile {
  LMSProfile (URI u);
  InitResponse getProfile (ProfileParams p);
}
```

This class is callable in the LPM; it defines the REST services implemented in the LMS to access to the information stored under the LMS Profile. These services are initiated by the INTUITEL system.

```
LMSProfile (URI u);
```

This method (called by the LPM) is the constructor for creating a structured variable in which the different values of the LMS are stored. The only *parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS.

```
InitResponse getProfile (ProfileParams p);
```

This method (called by the LMS) is used for returning the profile information the LMS. The response to this method returns the profile of the LMS according to the XML-based structure defined in the INTUITEL deliverable D1.1. The only *parameter* refers to:

- p: An XML fragment containing the profile parameters for the LMS, as defined in section 4.1 of D1.1.

2.1.1.2 Class Learners

```
Class Learners {
  Learners (URI u);
  LearnersPullResponse pullUpdate (PullParams p);
  pushUpdate (PushParams p);
  LearnersPollingResponse pushUpdateLearnerPolling (LearnerPollingParams
                    p);
}
```

The methods of this class are callable in the LPM or in the LMS and it defines the REST services to send the information related to the LO transitions performed by the learner in the LMS. These services are invoked either by the LMS which sends the updates to the INTUITEL system or by the INTUITEL system (in the Learner Polling scenario) when requests for updates from the LMS.

```
Learners (URI u);
```

This method (called by the LPM) is the constructor for creating a structured variable in which the different values of the Learners are stored. The only *parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS. In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
LearnersPullResponse pullUpdate (PullParams p);
```

This method (called by the LMS) sends a learner update message to INTUITEL and expects an immediate response directly containing all necessary data for the learner, meaning TUG and LORE data. The response to this method returns TUG and LORE items requested according to the structure defined in the INTUITEL deliverable D1.1. The *parameter* refers to:

- p: An XML fragment for the INTUITEL system as defined in section 4.2.1 of D1.1.

```
pushUpdate (PushParams p);
```

This method (called by the LMS) sends a learner update message to INTUITEL, which triggers the reasoning process. Contrary to the `pullUpdate` method, this one does not expect an immediate response and waits for the update message coming from the INTUITEL system. As such, there is no response to this method and the TUG and LORE messages are sent using the calls in sections 2.1.1.5 and 2.1.1.6. The *parameter* refers to:

- p: An XML fragment for the INTUITEL system as defined in section 4.2.2 of D1.1.

```
LearnersPollingResponse pushUpdateLearnerPolling (LearnerPollingParams p);
```

This method (called by the LPM) sends a message to the LMS asking for learner updates. The response to this is an XML-based message with the information related to the LOs visited by the learner. The only *parameter* refers to:

- p: An XML fragment for the INTUITEL system as defined in section 4.2.3 of D1.1.

2.1.1.3 Class Mapping

```
Class Mapping {
  Mapping (URI u);
  MappingResponse getMapping (MappingParams p);
}
```

The methods of this class are callable in the SLOM Repository or in the Editor and it defines the REST services to access the information stored about the LO Mappings by the LMS. These services are invoked by the INTUITEL System, e.g. the Editor.

```
Mapping (URI u);
```

This method (callable by the SLOM) creates a structured variable in which the different values of LO Mappings are stored. The only *parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS. In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
MappingResponse getMapping(MappingParams p);
```

This method (callable by the Editor) is used to retrieve the information related to the LO stored in the LMS. The only *parameter* refers to:

- p: An XML fragment for the INTUITEL system as defined in section 4.4 of D1.1.

2.1.1.4 Class Login

```
Class Login {
  Login (URI u);
  LoginResponse grantAccess (LoginParams p);
}
```

.

The methods of this class are callable in the LMS or in the SLOM, Editor or LPM. Thus, this class exposes REST services to grant access to the Content Creators through the LMS. The LMS will receive these services when the CC wants to create new content, i.e. accessing the INTUITEL Editor or Merger.

```
Login (URI u);
```

This method (callable by the LPM, the SLOM or the Editor) is the constructor, which creates a structured variable in which the information related to the authentication is stored. The only *parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS. In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
LoginResponse grantAccess (LoginParams p);
```

This method (called by the LMS) is used for logging in into the INTUITEL system by the content creator through the LMS. This method returns the grant access to the different LO IDs in the form of an XML fragment defined in the INTUITEL deliverable D1.1. The only *parameter* refers to:

- p: A data object (bean) containing the logging parameters, i.e. the user and the password, to access the INTUITEL System. This information will be sent using an XML fragment for the as defined in section 4.5 of D1.1.

2.1.1.5 Class TUG

```
Class TUG {
  TUG (URI u);
  TUGResponse sendTUGMessage (TUGReqParams p);
  delayedTUGResponse (TUGRespParams p);
}
```

The methods of this class are callable in the TUG sub-component of the LPM or in the LMS. Thus, this class exposes REST services to send TUG messages to the LMSs or back into the LPM. These services are invoked by the INTUITEL system.

```
TUG (URI u);
```

This method (callable by the LPM) is the constructor, which creates a structured variable in which the information related to the TUG message is stored. The only *parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS. In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
TUGResponse sendTUGMessage (TUGReqParams p);
```

This method (callable by the LPM) is used for sending TUG messages to the LMS connected. The *parameter* of this method is the following:

- p: An XML fragment for the INTUITEL system as defined in section 5 of D1.1.

There are two kinds of response although they follow the same XML-based structure already described in D1.1. Depending on the purpose of the message, the responses are:

- Immediate response: an XML-based message following the format described in D1.1 containing a list of acceptance (or error) items.
- Delayed response: if the TUG message sent by the system requires a response from the learner, the delayed response encloses the response of the learner. See the next method for more information on this response message.

```
delayedTUGResponse (TUGRespParams p);
```

This method (callable by the LMS) sends the delayed TUG response to the INTUITEL system. The *parameter* of this method is the following:

- p: An XML fragment for the INTUITEL system as defined in section 5 of D1.1. It contains the TUG response message, which propagates the information related to the LOs asked by previous TUG request messages for the INTUITEL system.

2.1.1.6 Class LORE

```
Class LORE {
  LORE (URI u);
  LOREResponse sendLOREMessage (LOREReqParams p);
}
```

The methods of this class are callable in the INTUITEL Engine. Thus, this class exposes REST services to send LORE messages to the LMSs and to the LPM. These services are invoked by the system and received by the LMS.

```
LORE (URI u);
```

This method (callable by the Engine) creates a structured variable in which the information related to the LORE message is stored. The only *Parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS (or the LPM). In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
LOREResponse sendLOREMessage (LOREReqParams p);
```

This method (callable by the Engine) is used for sending the LORE message to the LMS connected or to the LPM. The *Parameter* of this method is the following:

- p: An XML fragment for the INTUITEL system as defined in section 6 of D1.1.

Contrary to the TUG messaging, in this case there is only one kind of response, which is the immediate response. This response is an XML fragment following the format described in section 6 of D1.1 containing a list of acceptance (or error) items.

2.1.1.7 Class USE

```
Class USE {
  USE (URI u);
  USEResponse requestUSEInfo (USEReqParams p);
}
```

The methods of this class are callable in the USE sub-component of the LPM. Thus, this class exposes REST services to send USE messages to the INTUITEL system by the user of the LMS. However, these services are initiated by the INTUITEL system.

```
USE (URI u);
```

This method (callable by the LPM) creates a structured variable in which the information related to the USE message is stored. The only *Parameter* of the constructor is the following:

- u: An URI pointing to the address of the LMS. In the advanced scenario, the URI may refer also to a resource grouping a bunch of recipients like, e.g. a group of LMSs.

```
USEResponse requestUSEInfo (USEReqParams p);
```

This method (callable by the LPM) sends the USE message request to the LMS connected. The *parameter* of this method is the following:

- p: An XML fragment for the INTUITEL system as defined in section 7 of D1.1.

As the LORE messaging, there is only one kind of response, which is the immediate response. This is an XML fragment following the format described in section 7 of D1.1 containing a list of acceptance (or error) items. However, there are two kinds of requests depending on the information to be retrieved by the system:

- USE/performance: obtains the information related to the *performance* of the user, i.e. the learner with respect to specific LOs.
- USE/environment: obtains the information related to the *environment* in which the learner is located.

2.1.2 Internal API description

This API intends to describe the different methods, which are accessible only within the "Advanced" Communication Layer when it routes messages between different endpoints. As such, these methods are used for setting up the communication channels and for managing them.

2.1.2.1 Class MessageClient

```
Class MessageClient {
//Constructor
  MessageClient (String resourceId = null, String serverUrl = null);

//Starts connection
  void Connect ();

//Sending messages
  IntuitelMsg SendMsg (String recipient, String msgContent,
                       String recResourceId = null,
                       Boolean requestAck = false);
  IntuitelMsg SendMsgSync (String recipient, String msgContent,
                       String recResourceId = null, int timeout = 60000);

//Events
  void AddOnMsgEventListener (OnMsgListener subscriber)
  void RemoveOnMsgEventListener (OnMsgListener subscriber)
  interface OnMsgEventListener {
      void OnMessage (IntuitelMsg message);
  }

  void AddOnAckEventListener (OnAckEventListener subscriber)
  void RemoveOnAckEventListener (OnAckEventListener subscriber)
  interface OnAckEventListener {
      void OnAcknowledgement (String messageID, IID from);
  }
```

```
  void AddOnErrorEventListener (OnErrorListener subscriber)
  void RemoveOnErrorEventListener (OnErrorListener subscriber)
  interface OnErrorEventListener {
      void OnError (Exception exception);
  }

  void AddOnAuthErrorEventListener (OnAuthErrorListener subscriber)
  void RemoveOnAuthErrorEventListener (OnAuthErrorListener subscriber)
  interface OnAuthErrorEventListener {
      void OnAuthError (XmlElementexceptionXmlElement);
  }

  void AddOnDisconnectEventListener (OnMsgListener subscriber)
  void RemoveOnDisconnectEventListener (OnMsgListener subscriber)
  interface OnMDisconnectEventListener {
      void OnDisconnect ();
  }
}
```

2.1.2.1.1 Constructor

```
MessageClient (String resourceId = null, String serverUrl = null);
```

The constructor creates a MessageClient instance that is connected to the INTUITEL CL. The serverUrl is the only other external information that has to be known by the component. If the serverUrl is not provided (`null` value), the component, i.e. the CL, looks up in its directory the serverUrls needed by checking different Web services that can publish this information. The resourceId is the identifier for a specific instance of one component. For example, several instances of the INTUITEL Engine use different resourceIds. If then a message is sent to the Engine, the CL would automatically determine to which instance the message would be routed.

The *Parameters* of the constructor are the following:

- resourceId: A unique identifier for a specific instance of one account. Default: `null`.
- serverUrl: When another URL for connecting to the Communication Layer should be used than the officially known URLs, e.g. the main instance of the INTUITEL Engine, this can be specified by setting this parameter to the desired URL. Default: `null`.

2.1.2.1.2 Connect

```
void Connect ();
```

The connect method needs to be called after the events of the Message Client have been subscribed to, so that authentication errors can be handled. When a client is connected its status is set to available, when it disconnects from the servers its status is set to unavailable. This method has no parameters and returns no value.

2.1.2.1.3 Send message

```
IntuitelMsg SendMsg (String recipient, String msgContent,
                     String recResourceId = null,
                     Boolean requestAck = false);
```

This is the main message function to be used. If the caller wants to check if the message sent was received, the caller needs to observe/trace if an acknowledgement related to a sent messageId is received. As such, this method is asynchronous, the returning value is a pointer to the sent IntuitelMsg and the *Parameters* are the following:

- recipient: This is the target user to which the message should be sent.
- msgContent: This is the main message content to be routed to the target component, which can contain any kind of XML, JSON, other structured text or even unstructured text.
- recResourceId: This is the resource identifier, which identifies the programmatic object in case multiple objects were created that use the same user to communicate. Defaults to `null`, which means that the target is automatically determined or the message goes to all available instances.
- requestAck: This flag signalizes to the library that an acknowledgement message should be sent by the communication layer component in response to the receipt of the message and defaults to `false`.

2.1.2.1.4 Send sync message

```
IntuitelMsg SendMsgSync (String recipient, String msgContent,
                         String recResourceId = null, int timeout = 60000);
```

This is the synchronous version of the main message function just described. Its functionality is the same, but as synchronous it returns the direct response of the target component, and NOT a pointer to the sent message, when the response was received. Its *Parameters* are the following:

- recipient: This is the target user to which the message should be sent.
- msgContent: This is the main message content to be routed to the target component, which can contain any kind of XML, JSON, other structured text or even unstructured text.

- recResourceId: This is the resource identifier, which identifies the programmatic object in case multiple objects were created that use the same user to communicate. Defaults to `null`, which means that the target is automatically determined or the message goes to all available instances.
- timeout: The timeout is the time in milliseconds the MessageClient waits until it receives an answer. If the timeout is reached, the method returns a pointer to the sent IntuitelMsg instead of the received IntuitelMsg. The default timeout is 60 seconds.

2.1.2.1.5 Events related to OnMessage

```
void AddOnMsgEventListener (OnMsgListener subscriber)
void RemoveOnMsgEventListener (OnMsgListener subscriber)

interface OnMsgEventListener {
      void OnMessage (IntuitelMsg message);
}
```

The MessageClient instance that is connected to the Communication Layer receives messages for the logged in account. To be able to forward these messages, an object needs to be registered as an event listener to the MessageClient. This object has to implement the interface `OnMsgEventListener` and then be added to the list of objects that want to be informed about an incoming message. This can be done by calling the `AddOnMsgEventListener` and passing the object that implements `OnMsgEventListener` as parameter.

`OnMsgListener` is the interface to be implemented by any class that wants to be able to receive IntuitelMsgs delivered by the MessageClient class. This is connected to the Communication Layer servers. The implemented method returns no values and the *parameter* of the implemented method is the following:

- message: This is the message wrapper and contains the message content.

2.1.2.1.6 Events related to OnAcknowlegement

```
void AddOnAckEventListener (OnAckEventListener subscriber)
void RemoveOnAckEventListener (OnAckEventListener subscriber)

interface OnAckEventListener {
      void OnAcknowledgement (String messageID, IID from);
}
```

The `OnAcknowledgement` method is called on all event subscribers implementing the `OnAckEventListener` interface that were added to the list of subscribers via `AddOnAckEventListener` when the requested acknowledgement for a sent message comes in. The acknowledgement will not trigger the signalling of the `OnMessage`.

`OnAckListener` is the interface to be implemented by any class that wants to be able to receive acknowledgment notifications for sent messages requesting an acknowledgment answer delivered by the MessageClient class. This is connected to the Communication Layer servers. The implemented method returns no values and the *parameters* of the implemented method are the following:

- messageID: The ID of the message that was acknowledged by the receiver.
- from: An IID (INTUITEL ID) from whom the message acknowledgement was sent.

2.1.2.1.7 Events related to OnError

```
void AddOnErrorEventListener (OnErrorListener subscriber)
void RemoveOnErrorEventListener (OnErrorListener subscriber)

interface OnErrorEventListener {
      void OnError (Exception exception);
}
```

The `OnError` method is called on all event subscribers implementing the `OnErrorEventListener` interface that were added to the list of subscribers via `AddOnErrorEventListener` when an error occurs.

`OnErrorEventListener` is the interface to be implemented by any class that wants to be able to receive exceptions when something unexpected happens. The implemented method returns no values and the *parameter* of the implemented method is the following:

- exception: The Exception object describing the Error, possibly also giving additional information as to how to handle the problem.

2.1.2.1.8 Events related to OnAuthError

```
void AddOnAuthErrorEventListener (OnAuthErrorListener subscriber)
void RemoveOnAuthErrorEventListener (OnAuthErrorListener subscriber)

interface OnAuthErrorEventListener {
      void OnAuthError (XmlElementexceptionXmlElement);
}
```

The `OnAuthError` method is called on all event subscribers implementing the `OnAuthErrorEventListener` interface that were added to the list of subscribers via `AddOnAuthErrorEventListener` when an authentication error occurs. This should happen only right after the connect method on the MessageClient was called.

`OnAuthErrorEventListener` is the interface to be implemented by any class that wants to be able to receive authentication exceptions when the authentication information, which consists of username and password, are not valid. The implemented method returns no values and the *parameter* of the implemented method is the following:

- exceptionXmlElement: The received XMLElement describing the Error, possibly also giving additional information as to how to handle the problem.

2.1.2.1.9 Events related to OnDisconnect

```
void AddOnDisconnectEventListener (OnMsgListener subscriber)
void RemoveOnDisconnectEventListener (OnMsgListener subscriber)

interface OnMDisconnectEventListener {
      void OnDisconnect ();
}
```

The `OnDisconnect` method is called on all event subscribers implementing the `OnDisconnectEventListener` interface that were added to the list of subscribers via `AddOnDisconnectEventListener` when an authentication error occurs.

`OnDisconnectEventListener` is the interface to be implemented by any class that wants to be able to be notified when the connection was disconnected, for example, when the server is unavailable or the internet connection is down. The implemented method returns no values and has no *parameters*.

2.1.2.2 Class IntuitelMsg

```
IntuitelMsg (XMLMessage xmlMessage);
```

The other components will never need this constructor, as the IntuitelMsg will be created in the SendMsg and SendMsgSync methods. Incoming XMLMessages will always be wrapped into an IntuitelMsg by the event that delivers the message. The *Parameter* is the following:

- xmlMessage: The wrapped library's implementation of the XML Message object implementation of INTUITEL. The other components will not need this constructor as it is only used by the MessageClient.

```
String getMessageBody ();
```

This will contain the Message body sent to the component. Usually, the component can de-serialize the string with its self-defined XSD to generate an object to handle the message. This method uses no *parameters* and returns the body of the message.

```
bytes[] getFileAttachment (String key = null);
```

This method returns the file attachment that can be located with the key parameter. The return type could be changed if there is a way to return the file with increased performance than returning the byte array. The return value is the binary file attached to the message, or null if there is no binary attachment and the *parameter* is the following:

- key: The identifier of the attachment, if null is passed the first attachment is returned.

```
void setFileAttachment (DataHandlerdataHandler, String key = null);
```

This method stores a file attachment to the current message. There is no return value and the *parameters* are the following:

- dataHandler: A data handler that represents the bytes of the attached file.
- key: The key, under which the file will be appended to the message. If null is passed to this method, a random key will be generated. Default: `null`.

```
IID getReceiver ();
```

This method returns the IID (INTUITEL ID) of the receiver.

```
IID getSender ();
```

This method returns the IID (INTUITEL ID) of the sender.

```
String getMessageId ();
```

This method returns the ID of the message.

```
String getThreadId ();
```

This method returns the ID of the message exchange thread. This is used to identify if this message is an answer to another message or is used by the receiver to answer a message.

```
DateTime getReceived ();
```

This method returns the time stamp when the message was received.

```
DateTime getSent ();
```

This method returns time stamp when the message was sent. This will contain the Message body sent to the component. Usually, the component can de-serialize the string with its self-defined XSD to generate an object to handle the message.

```
boolean getAcknowlegementWanted ();
```

This method returns true if an acknowledgement message is wanted, false otherwise.

```
String setAcknowledgementWanted (boolean acknowledgementWanted);
```

This method sets the flag when the receiver should answer with an acknowledgment to the message sent. The *parameter* is a simple Boolean variable. This will be set by the MessageClient.

```
DateTime getAcknowlegementReceivedTime ();
```

This method returns the time when the acknowledgement was received or `null` if no acknowledgement was received.

```
String serialize ();
```

This method returns the necessary XML metadata to transfer over XMPP including the message body as a String. This is used internally by MessageClient when it is implemented over XMPP protocol.

```
IntuitelMsg deserialize (String serializedIntuitelMsg);
```

This method returns the necessary XML metadata to transfer over XMPP including the message body as a String. This is used internally by MessageClient when it is implemented over XMPP protocol.

2.1.2.3 Class IID

This class identifies the INTUITEL message that is exchanged between the different components. This class consists of serverUrl and resource identifier. As the serverUrl is known, only resource identifier is needed.

```
IID (String resourceId = null, String serverUrl = null);
```

The constructor creates an IID instance. The *parameters* are the following:

- resourceId: A unique identifier for a specific instance of one component. It is `null`, when not specified.
- serverUrl: If another URL than the known URLs for the INTUITEL Communication Layer should be used, it can be set as a parameter. Default: `null`.

```
void toString ();
```

This method returns the String value, e.g. resourceId@intuitel.eu/instanceid

2.2 "Basic" Communication Layer Development

2.2.1 Installation

The "Basic" Communication Layer is accessible at the following URL address:

http://95.211.177.222/IntuitelCommunicationManager/CommunicationManagerService.svc/rest

To ease the development of client applications, it is implemented using REST, SOAP and WCF (Windows Communication Foundation) so the client is free to choice the best option for his development. The following URL points to the WDSL location:

http://95.211.177.222/IntuitelCommunicationManager/CommunicationManagerService.svc/basic?wdsl

The "Basic" Communication Layer can be deployed inside a Web Server or can run within the included host application. This host application can be configured selecting the URL endpoints, multiple binding services. Please, refer to section 2.2.7 for a more detailed description of this host application.

2.2.2 Interfaces Description

```
/// <summary>
/// Generic process message, launch a document analyzer, transform it and
route it to targeted service.  Accepts all types of message.
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string Message(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a LoMappingRequest or a
LoMappingResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string LoMapping(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a LmsProfileRequest or a
LmsProfileResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string LmsProfile(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a LearnerUpdateRequest or a
LearnerUpdateResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string LearnerUpdate(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a UseRequest or a
UseResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string Use(Stream request)
```

```
/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a RecommendationRequest or
a RecommendationResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string Recommendation(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a LoreRequest or a
LoreResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string Lore(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a TugRequest or a
TugResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string Tug(Stream request)

/// <summary>
/// Named method to process message, launch a document analyzer, transform
it and route it.  Check if the messages matches a TugDelayedResponse
/// </summary>
/// <param name="request">Xml payload to process</param>
/// <returns>Response from the target service</returns>
    public string TugDelayedResponse(Stream request)
```

2.2.3 Behaviour

When a message is received in the Message Interface, it is processed to match it against one of the message types (via its properties). In the case that none of the message types matches the XML content, then an error response is sent.

If the type of the message has specified a XSLT Transformation, this is performed. Then, as a final step, the system looks for the destination to route the message.

Each message type present in the system will have a specific route for delivering the messages of that particular type. The selection of the appropriate route depends upon the match of the following criteria described in the configuration: the XPath, the namespace, the DTD or the Schema. If, the message received, does not specify any of these types, then the selected route is the one with those parameters empty.

In any case, the message can overwrite the parameters provided by the exposed method. In this case, this has to be done in the parameters of the call. In the case an error occurs, the system will send the response with an error description.

2.2.4 Configuration

2.2.4.1 Configuration file location

The configuration file is set trough service's web.config file, located at the web application container folder, in the section appSettings:

```
<appSettings>
  <add key="CONFIG_FILE_PATH"
                    value="C:\Intuitel\IntuitelCommunicationManager\Intuit
                    el.config"/>
</appSettings>
```

2.2.4.2 File structure

2.2.4.2.1 MessageTypes

This section specifies all the types of message that can be processed:

```
<IntuitelMessageType>
  <Name>UseRequest</Name>
  <XPathRecognition>/intuilpm:INTUITEL/intuilpm:UsePerf
                    </XPathRecognition>
  <NamespaceRecognition>http://www.intuitel.eu/public/intui_DMLPM.xsd
                    </NamespaceRecognition>
</IntuitelMessageType>
```

- Name (M): Name of the message type allowing its identification in the configuration fields
- XPathRecognition (O): XPath to match to the message type
- NamespaceRecognition (O): Namespace contained in the message
- SchemaRecognition (O): Schema against which the message is validated
- DtdRecognition (O): Document Type Definition to search for the message
- XsltTransformation (O): Full path to the XSLT file to apply transformation to the XML message received
- XPathMessageIdValue (O): XPath pointing to the node value containing the Id message

2.2.4.2.2 Components

This section specifies all the components plugged in the system, including its provided services and end point URL:

```
<IntuitelComponent>
  <Name>IntuitelEngine</Name>
  <ConnectionType>Rest</ConnectionType>

                    <EndPoint>http://95.211.177.222/IntuitelTestComponents
                    /IntuitelEngineService.svc/basic?wdsl</EndPoint>
  <ServiceName>IIntuitelEngineService</ServiceName>
  <MethodList>
    <IntuitelComponentService>
      <Name>Recommendation</Name>
      <ParamList>
        <IntuitelServiceMethodParameter>
          <Name>requestXml</Name>
          <Type>Xml</Type>
          <Value>[message]</Value>
          <ValueIsConstant>false</ValueIsConstant>
        </IntuitelServiceMethodParameter>
      </ParamList>
      <ReturnType>Xml</ReturnType>
    </IntuitelComponentService>
  </MethodList>
</IntuitelComponent>
```

- Name (M): Name of the component allowing its identification in the configuration fields
- ConnectionType (M): Component's connection type: REST, SOAP or WCF
- EndPoint (M): Endpoint URI of the service. In WCF this is the WDSL descriptor file used to generate on the fly the client classes
- ServiceName (O): Service name. In WCF communication service, defines the contract interface name implemented
- InitMethodCall (O): Method comma separated list to call before any service call
- CloseMethodCall (O): Method comma separated list to call when the application service is closed
- MethodList (O): Service method list supplied by the component

2.2.4.2.3 Method List

The Method List describes all the services supplied by the component IntuitelComponentService.

```
<MethodList>
  <IntuitelComponentService>
    <Name>Recommendation</Name>
    <ParamList>
      <IntuitelServiceMethodParameter>
        <Name>requestXml</Name>
        <Type>Xml</Type>
        <Value>[message]</Value>
        <ValueIsConstant>false</ValueIsConstant>
      </IntuitelServiceMethodParameter>
    </ParamList>
    <ReturnType>Xml</ReturnType>
  </IntuitelComponentService>
</MethodList>
```

- Name (M): Component's service name
- ReturnType (M): Response's type. XML, String, void. If XML is received then a stream response is expected, else a common string. If void, the response is not used.
- ParamList: Parameters needed to perform a call to this component's service. Structure:
    - Name (M): Param's name.
    - Type (M): Param's type.
    - Value (O): Value to send with the call. Values in brackets are calculated. Could be an XPath value to extract from the request.
    - ValueIsConstant (O): Value will not be calculated.

2.2.4.2.4 Routes

This section describes all the routes for each message type. It is possible to define multiple routes for each message type and the system selects the correct way by recognising the configuration properties. If a route does not match the message content, a route with empty route is selected for that message type.

```
<Routes>
  <IntuitelRoute>
    <SourceMessageType>LearnerUpdateRequest</SourceMessageType>
    <DestinationComponentName>LPM</DestinationComponentName>
    <MethodNameToBeCall>LearnerUpdate</MethodNameToBeCall>
    <ParamList />
  </IntuitelRoute>
</Routes>
```

- SourceMessageType (M): Message type which will be routed
- DestinationComponentName (M): Destination component of the message
- MethodNameToBeCall (M): Method of the component's method list to be call
- AsynchronousCall (O): Yes/No. Indicates if the communication layer has to wait for a response or not. By default the value is No
- XPathRecognition (O): XPath to match to the route. Used in case of multiple routes for a message type
- NamespaceRecognition (O): Namespace contained in the message to select a route. Used in case of multiple routes for a message type
- SchemaRecognition (O): Schema which against is validate the message to select a route. Used in case of multiple routes for a message type
- DtdRecognition (O): Document Type Definition to search in message to select a route. Used in case of multiple routes for a message type
- XsltTransformation (O): Full path to the XSLT file to apply transformation to the xml message to send

2.2.4.2.5 DefaultNamespaces

This section is used for helping in the XPath recognition as it includes the default namespaces with their prefixes.

```
<DefaultNamespaces>
  <IntuitelDefaultNamespace>
    <Prefix>intuilms</Prefix>
    <Value>http://www.intuitel.eu/public/intui_DMLMS.xsd</Value>
  </IntuitelDefaultNamespace>
</DefaultNamespaces>
```

- Prefix (M): The prefix to be used.
- Value (O): The namespace represented.

2.2.5 Accepted Message Types

Although the system is prepared to work with any message type, the current system's installation accepts the message types described in the following sections.

2.2.5.1 LMS Learner Update

2.2.5.1.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

  <intuilms:Learner mId="12345678-1234-abcd-ef12-123456789012"
                     uId="jmb0001" loId="LO4711" time="1362575466"/>
</intuilms:INTUITEL>
```

2.2.5.1.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">

  <intuilpm:Learner mId="12345678-1234-abcd-ef12-123456789012"
                     uId="jmb0001">
    <intuilpm:Lore uId="jmb0001" mId="12345678-1234-abcd-ef12-
                     123456789013">
      <intuilpm:LorePrio loId="LO4711" value="42"/>
      <intuilpm:LorePrio loId="LO4712" value="50"/>
    </intuilpm:Lore>
    <intuilpm:Tug uId="jmb0001" mId="12345678-1234-abcd-ef12-123456789014">
      <intuilpm:MType>1</intuilpm:MType>
      <intuilpm:MData>Good Morning, dear Learner!</intuilpm:MData>
    </intuilpm:Tug>
  </intuilpm:Learner>
 </intuilpm:INTUITEL>
```

2.2.5.2 LMS Profile Message

2.2.5.2.1 Request

```
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:LmsProfile mId="12345678-1234-abcd-ef12-123456789012"/>
</intuilpm:INTUITEL>
```

2.2.5.2.2 Response

```
"OK"
```

2.2.5.3 Authentication

2.2.5.3.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:Authentication uId="jmb0001" mId="12345678-1234-abcd-ef12-
                     123456789012">
    <intuilpm:Pass>Europe</intuilpm:Pass>
  </intuilpm:Authentication>
</intuilpm:INTUITEL>
```

2.2.5.3.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

  <intuilms:Authentication uId="jmb0001" mId="12345678-1234-abcd-ef12-
                     123456789012" status="OK">
    <intuilms:LoPerm loId="LO4711"/>
    <intuilms:LoPerm loId="LO4712"/>
  </intuilms:Authentication>
</intuilms:INTUITEL>
```

2.2.5.4 LearnerUpdate Polling

2.2.5.4.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">

  <intuilpm:Learners mId="12345678-1234-abcd-ef12-123456789012"/>
</intuilpm:INTUITEL>
```

2.2.5.4.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

  <intuilms:Learners mId="12345678-1234-abcd-ef12-123456789012">
    <intuilms:Learner uId="jmb0001">
      <intuilms:VisitedLo loId="LO4711" time="1362575466"/>
    </intuilms:Learner>
    <intuilms:Learner uId="mean0001">
      <intuilms:VisitedLo loId="LO4711" time="1362575466"/>
      <intuilms:VisitedLo loId="LO4712" time="1362575466"/>
    </intuilms:Learner>
  </intuilms:Learners>
</intuilms:INTUITEL>
```

2.2.5.5 LoMapping

2.2.5.5.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:LoMapping mId="12345678-1234-abcd-ef12-123456789013">
    <intuilpm:Data name="loName" value="Math Intro, Test 1"/>
  </intuilpm:LoMapping>
</intuilpm:INTUITEL>
```

2.2.5.5.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

  <intuilms:LoMapping mId="12345678-1234-abcd-ef12-123456789012">
    <intuilms:Data name="loId" value="LO4712"/>
    <intuilms:Data name="loName" value="Math Intro, Test 1"/>
    <intuilms:Data name="hasParent" value="Course15"/>
    <intuilms:Data name="loType" value="test"/>
    <intuilms:Data name="learningTime" value="00:10:00"/>
  </intuilms:LoMapping>
</intuilms:INTUITEL>
```

2.2.5.6 Lore

2.2.5.6.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:Lore uId="jmb0001" mId="12345678-1234-abcd-ef12-123456789012">
    <intuilpm:LorePrio loId="LO4711" value="42"/>
    <intuilpm:LorePrio loId="LO4712" value="50"/>
  </intuilpm:Lore>
</intuilpm:INTUITEL>
```

2.2.5.6.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

 <intuilms:Lore uId="jmb0001" mId="12345678-1234-abcd-ef12-123456789012"
                     retVal="OK"/>
  </intuilms:INTUITEL>
```

2.2.5.7 Tug

2.2.5.7.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:Tug uId="jmb0001" mId="12345678-1234-abcd-ef12-123456789012">
    <intuilpm:MType>1</intuilpm:MType>
    <intuilpm:MData>Good Morning, dear Learner!</intuilpm:MData>
  </intuilpm:Tug>
</intuilpm:INTUITEL>
```

2.2.5.7.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">

 <intuilms:Tug uId="jmb0001" mId="12345678-1234-abcd-ef12-123456789012"
                     retVal="OK"/>
  </intuilms:INTUITEL>
```

2.2.5.8 Use Environmental

2.2.5.8.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:UseEnv uId="jmb0001" mId="12345678-1234-abcd-ef12-
                     123456789012">
    <intuilpm:Data name="lName"/>
  </intuilpm:UseEnv>
</intuilpm:INTUITEL>
```

2.2.5.8.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">
                     <intuilms:UseEnv uId="jmb0001" mId="12345678-1234-
                     abcd-ef12-123456789012" retVal="OK">
    <intuilms:Data name="lName" value="José Manuel Barroso"/>
    <intuilms:Data name="lGender" value="M"/>
    <intuilms:Data name="lAge" value="56"/>
    <intuilms:Data name="eTime" value="12:00:00"/>
    <intuilms:Data name="dType" value="phone"/>
  </intuilms:UseEnv>
</intuilms:INTUITEL>
```

2.2.5.9 Use Performance

2.2.5.9.1 Request

```
<?xml version="1.0" encoding="UTF-8"?>
<intuilpm:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLPM.xsd
                     http://www.intuitel.de/public/intui_DMLPM.xsd"
 xmlns:intuilpm="http://www.intuitel.eu/public/intui_DMLPM.xsd">
  <intuilpm:UsePerf uId="jmb0001" mId="12345678-1234-abcd-ef12-
                     123456789012">
    <intuilpm:LoPerf loId="LO4711"/>
    <intuilpm:LoPerf loId="LO4712"/>
  </intuilpm:UsePerf>
</intuilpm:INTUITEL>
```

2.2.5.9.2 Response

```
<?xml version="1.0" encoding="UTF-8"?>

<intuilms:INTUITEL xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.intuitel.eu/public/intui_DMLMS.xsd
                     http://www.intuitel.de/public/intui_DMLMS.xsd"
 xmlns:intuilms="http://www.intuitel.eu/public/intui_DMLMS.xsd">
                     <intuilms:UsePerf uId="jmb0001" mId="12345678-1234-
                     abcd-ef12-123456789012">
    <intuilms:LoPerf loId="LO4711">
      <intuilms:Score type="completion" value="100"/>
```

```
      <intuilms:Score type="grade" value="1"/>
      <intuilms:Score type="seenPercentage" value="100"/>
    </intuilms:LoPerf>
  </intuilms:UsePerf>
</intuilms:INTUITEL>
```

2.2.6 REST Test Sample code

```
String learnerUpdateRequest = "<?xml version=\"1.0\" encoding=\"UTF-
                    8\"?><intuilms:INTUITEL
                    xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
                    instance\"
                    xsi:schemaLocation=\"http://www.intuitel.eu/public/int
                    ui_DMLMS.xsd
                    http://www.intuitel.de/public/intui_DMLMS.xsd\"
                    xmlns:intuilms=\"http://www.intuitel.eu/public/intui_D
                    MLMS.xsd\"> <intuilms:Learner mId=\"12345678-1234-
                    abcd-ef12-123456789012\" uId=\"jmb0001\"
                    loId=\"LO4711\" time=\"1362575466\"/>
                    </intuilms:INTUITEL>";
ClientConfig config = new DefaultClientConfig();
     Client client = Client.create(config);
     WebResource service = client.resource(getBaseURI());
    ClientResponse response =
                    service.path("LearnerUpdate").type(MediaType.APPLICATI
                    ON_FORM_URLENCODED)
            .post(ClientResponse.class, learnerUpdateRequest);

     System.out.println("Form response :" +
                    response.getEntity(String.class));
```

2.2.7 Host Application

Along with the "Basic" Communication Layer a configurable host application is included. In this configuration file, it is possible to set the endpoints, bindings and behaviours of the "Basic" Communication Layer Service:

```
<bindings>
   <webHttpBinding>
    <binding name="webHttpBindingConfiguration" receiveTimeout="00:10:00"
                    sendTimeout="00:10:00"
                    maxReceivedMessageSize="2147483647">
     <readerQuotas maxDepth="2147483647"
                    maxStringContentLength="2147483647"
                    maxArrayLength="2147483647"
                    maxBytesPerRead="2147483647"
                    maxNameTableCharCount="2147483647" />
```

```
    </binding>
   </webHttpBinding>
   <wsHttpBinding>
    <binding name="WSHttpBindingConfiguration" receiveTimeout="00:10:00"
                    sendTimeout="00:10:00"
                    maxReceivedMessageSize="2147483647">
     <readerQuotas maxDepth="2147483647"
                    maxStringContentLength="2147483647"
                    maxArrayLength="2147483647"
                    maxBytesPerRead="2147483647"
                    maxNameTableCharCount="2147483647" />
    </binding>
   </wsHttpBinding>
   <basicHttpBinding>
    <binding name="BasicHTTPConfiguration" receiveTimeout="00:10:00"
                    sendTimeout="00:10:00" maxBufferSize="2147483647"
                    maxReceivedMessageSize="2147483647">
     <readerQuotas maxDepth="2147483647"
                    maxStringContentLength="2147483647"
                    maxArrayLength="2147483647"
                    maxBytesPerRead="2147483647"
                    maxNameTableCharCount="2147483647" />
    </binding>
   </basicHttpBinding>
  </bindings>
  <services>
   <service behaviorConfiguration="RestBehavior"
                    name="Intuitel.BasicCommunicationManager.Communication
                    ManagerService">
    <endpoint behaviorConfiguration="Web" binding="webHttpBinding"
                    address="rest"

                    contract="Intuitel.BasicCommunicationManager.ICommunic
                    ationManagerService"
                    bindingConfiguration="webHttpBindingConfiguration"
     name="WebHttpBinding" />
    <endpoint binding="mexHttpBinding" name="mex"
                    contract="IMetadataExchange" address="mex" />
    <endpoint binding="basicHttpBinding" name="basicHttpBinding"
                    address="basic"

                    contract="Intuitel.BasicCommunicationManager.ICommunic
                    ationManagerService" />
    <host>
     <baseAddresses>
      <add baseAddress="http://localhost:8181/CommunicationManagerService"
                    />
     </baseAddresses>
    </host>
   </service>
```

```
  </services>
  <behaviors>
   <endpointBehaviors>
    <behavior name="Web">
     <webHttp />
    </behavior>
   </endpointBehaviors>
   <serviceBehaviors>
    <behavior
                    name="Intuitel.BasicCommunicationManager.Communication
                    ManagerServiceBehavior">
     <serviceMetadata httpGetEnabled="true"
                    httpGetUrl="http://localhost:8181/CommunicationManager
                    Service"/>
     <serviceDebug includeExceptionDetailInFaults="false" />
    </behavior>
    <behavior name="RestBehavior">
     <serviceMetadata httpGetEnabled="true"
                    httpGetUrl="http://localhost:8181/CommunicationManager
                    Service" />
     <serviceDebug />
    </behavior>
   </serviceBehaviors>
  </behaviors>
```

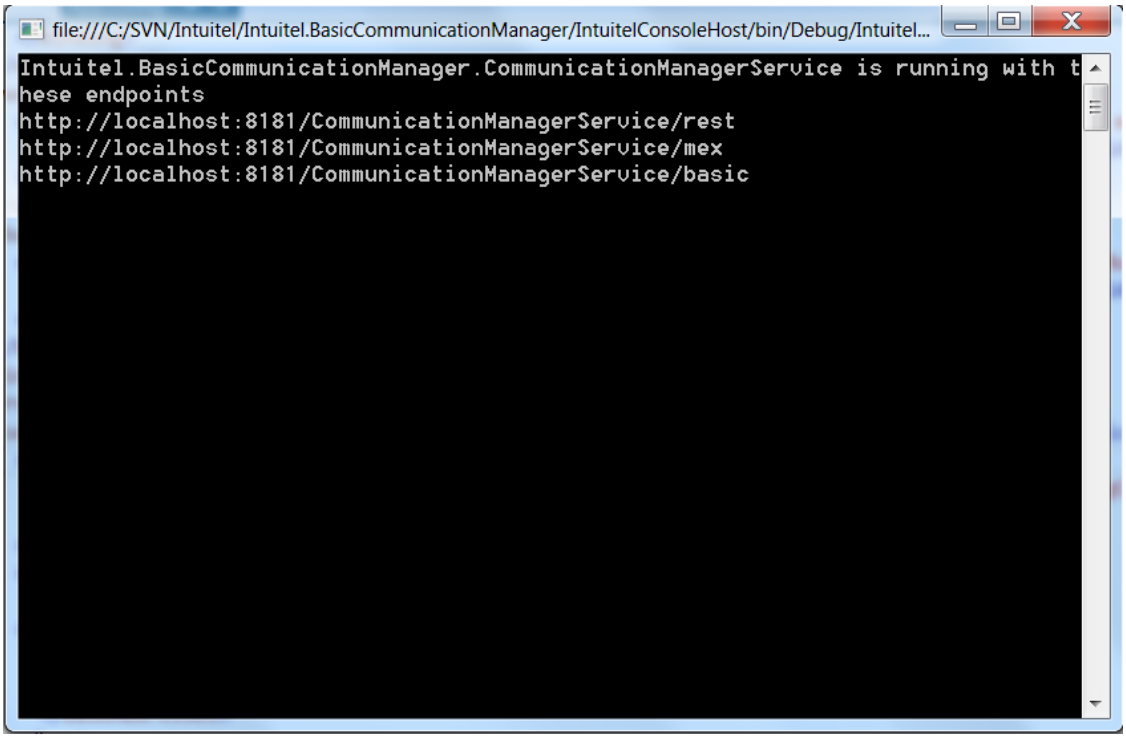

Figure 8: Screenshot of the "Basic" Communication Layer Host Application

2.3 "Advanced" Communication Layer Development

The development of the "Advanced" Communication Layer has not yet started as the development team focused on having tangible results first, that could be tested and presented to the rest of the INTUITEL Consortium, rather than partially develop both solutions and not even enable the tests which are described in the following chapter.

In any case, task 3.3 is still a live task and will run for three more months in which the development of the "Advanced" CL will be completely performed and tested.

3 Communication Layer Tests

3.1 Introduction

With the goal of ensuring completeness and reliability of the CL, a testing plan has supported the software development stage. Furthermore, the methodology adopted at the testing stage also checks operability in cross-domain situations. Since the REST interface does not change in the basic and advanced versions of the CL, the same test suite is valid for both versions.

The testing plan is based on the development of unit tests, which are focused on the validation and completeness of the exchanged XML messages. In order to ensure independent testing, and also operability in cross-domain situations, the test suite and the CL were developed by developer teams from different institutions. Additionally, the testing plan designed in the scope of T3.3 is also valid for T3.4, devoted to verify the connection between different subsystems of the INTUITEL system.

PHP is the programming language used for the implementation of testing suite. In particular, the PHPUnit[4] libraries have been used. According to the provided licence terms, the redistribution and use of PHP unit are permitted under some conditions that are satisfied by the INTUITEL project[5].

3.2 Testing methodology

3.2.1 Participant Roles

Different developer teams contributed to this testing task. For the sake of clarity, this document uses the following terminology:

- CL developers: This refers to the developers of the communication layer, both in the basic and the advanced versions.
- Testers: This refers to the developers of the testing suite.
- LMS developers: This refers to the developers of the different INTUITEL subsystems, including LMSs.

[4] http://phpunit.de
[5] https://github.com/sebastianbergmann/phpunit/blob/master/LICENSE

3.2.2 Requirements

Since the test suite and the CL were developed by developer teams from different institutions, it was important to establish a well-defined set of requirements that should be covered by the CL. Such specification is given in D1.1, and collects all the REST message formats that should be supported by the INTUITEL subsystems.

As the CL serves as the communication middleware for any communication between INTUITEL subsystems, all the defined message types are tested within the created testing suite. That is:

- <INTUITELEndPoint>/lmsprofile
- <INTUITELEndPoint>/learners
- <INTUITELEndPoint>/mapping
- <INTUITELEndPoint>/login
- <INTUITELEndPoint>/TUG
- <INTUITELEndPoint>/LORE
- <INTUITELEndPoint>/USE/performance
- <INTUITELEndPoint>/USE/environment

Therefore, the CL developers only needed to publish the service and provide the testers with the corresponding URL.

3.2.3 Planned procedure

Since the beginning of the development, the testers and the CL developers have been coordinated and they have planned a testing procedure that ensures an iterative development that receives the feedback from the testing phase and improves the final implementation of the communication layer. The testing stages are as follows:

- At an early stage of development, the CL developers send the documentation to the testers, including the basic architectural principles and the supported messages types. The specific architectural details may change, but the message types should not change. This ensures a parallel development of the CL and the testing suite.
- The CL developers and the testers work in parallel and elaborate the CL and test suite, respectively.
- Once the CL is publicly[6] available and its URL has been communicated to the testers, the testing suite runs all the unit tests. This stage results in a report (a log file) with the detail of the testing procedure, which will be sent to the CL developers.
- The CL developers review the completeness of the testing suite and suggest further unit tests, that will be implemented by the testers and include in future iterations of the testing procedure.

[6] That is, the server can receive messages from the Internet, under some pre-established security considerations.

3.2.4 Result reports

As the goal of the testing task is to improve the code reliability, it is important to establish a solid procedure that communicates the result reports to the CL developers and LMS developers, so they can solve the detected issues. The established procedure, depicted in Figure 9, works as follows:

The test suite is periodically executed, generating a results report. Such report is first locally stored, and also becomes available via web. Additionally, the test server can send an email to the CL developers (or LMS developers) with the last generated report. This way, the developers are instantly reported with the last test results, at the time they have access to the log archive.

The complete documentation of the testing suite contains the log archive, an up-to-date list of all developed tests, and the source code of the test suite.

It is available at http://tel.unir.net/intuitel-tests/.

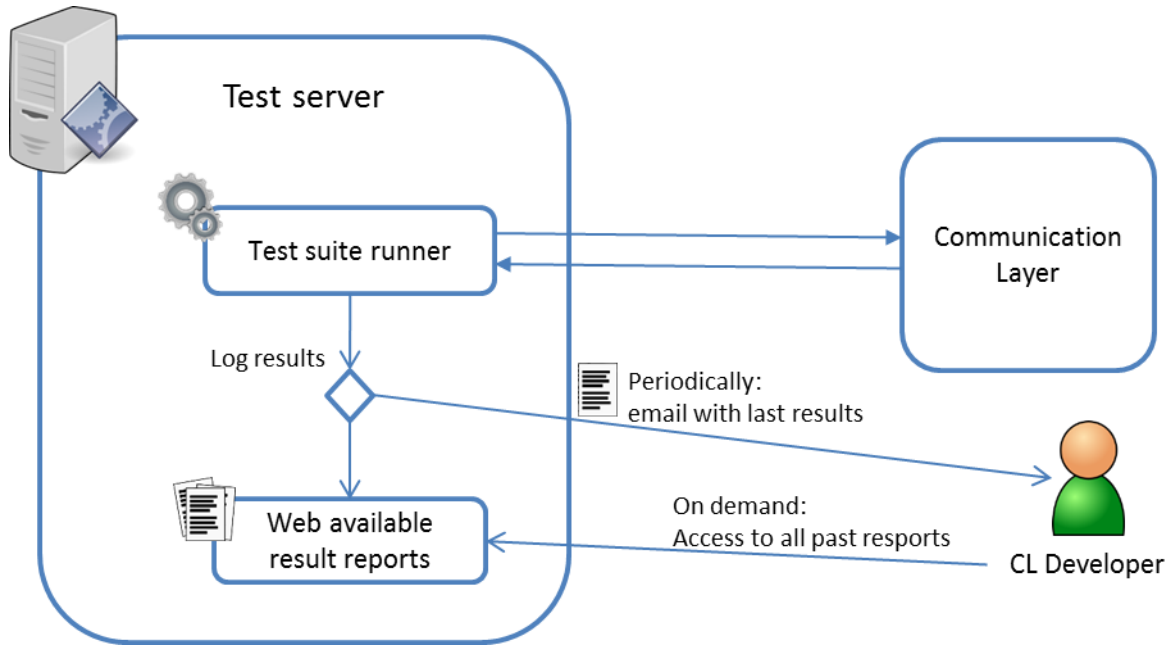

Figure 9: Communication of result reports

The result reports are automatically generated by the testing suite, and contain the information summarized in Table 3.

Item	Description
Number of executed test	Total number of executed tests
ACK average time	Average time waiting for the HTTP status code
Response average time	Average time waiting for the response, in those cases where the complete response comes after the HTTP status code
Successful tests	Number of test that obtained the expected response
Failed tests	Number of tests that did not obtain the expected response

Table 3: General results report

In addition, each of the failed tests will result in a case specific report with the information of Table 4. The description of successful test will be available via in the documentation.

Item	Description
Test case description	A textual description of the action that was being executed, including the complete request body, and the URL being tested.
Expected result	The output that was expected from the REST service
Obtained result	The actual output obtained from the REST service
Further details	If applicable, a textual description of any other detail that could be useful for the identification of the problem in the specific test case

Table 4: Case specific result report

3.3 Developed Tests

This section offers a list of the tests developed in the first version of the testing suite. Thus, Table 5 shows the test cases developed for the testing of the LMS Profile Message type. The test suite for the rest of the message types follows the same scheme. That is, the tests are the same, with the minor corrections required to adjust the test cases to the message type (e.g. the name of the XML elements is different). An up-to-date list is available at the test suite documentation web page.

Test name	Description	Expected result
LMS_Profile_Message_001.1	Send valid-simple request and check the HTTP response code	HTTP Status: 200
LMS_Profile_Message_001.2	Send valid-simple request and check the well formation of the response	Well-formed XML received in the payload
LMS_Profile_Message_001.3	Send valid-simple request and check the validity of the response	Valid XML received in the payload
LMS_Profile_Message_004	Send valid request, with different content-type in POST header	HTTP Status: 4XX (bad request)
LMS_Profile_Message_005	Send request with wrong root element name (LmsProfile)	HTTP Status: 4XX (bad request)
LMS_Profile_Message_006	Send request with wrong root element name (INTUITEL)	HTTP Status: 4XX (bad request)
LMS_Profile_Message_007	Send request with wrong attribute name (mId)	HTTP Status: 4XX (bad request)
LMS_Profile_Message_008	Send request with empty attribute value (mId)	HTTP Status: 4XX (bad request)
LMS_Profile_Message_009	Send request with empty payload (no XML sent)	HTTP Status: 4XX (bad request)

Table 5: Test suite for LMS Profile Message type

4 Appendix: Technology Comparison

4.1 Major Design Decisions

The following is a list of design decisions ascribable to the Communication Layer:

- It should work asynchronously.
- It has to be unobtrusive, e.g. should affect the code of the LMS as less as possible.
- It should be easily usable as a service by any of the INTUITEL components.
- It also has to facilitate the routing of structured text messages (like XML or JSON) as well as binary files.
- In the "Advanced" version it should have a very high throughput of messages, as all components will use those to call services in other components.
- In the "Advanced" version it must provide secure communication.
- In the "Advanced" version it also has to buffer messages and binary files in the following cases:
    - A receiver is not able to receive the message / file.
    - The receiver is offline, meaning the LMS is down.
    - The receiver rejected the message, like the TUG response PAUSE: message should be resent.

4.2 Technology Comparison and Selection

This subsection will compare existing technologies for the Communication Layer component, including REST and SOAP (WSDL) Web services, XMPP, ActiveMQ and TIE SmartBridge. These technologies are further discussed as well as the rationale behind deciding for or against these solutions for exploiting them in Communication Layer.

However, prior the comparison, a set of parameters for the selection criteria will be introduced. Sections 4.2.1.1 and 4.2.1.2 define the selection criteria for the technology comparison.

4.2.1 Definition of the Selection Criteria

The selection criteria for selecting the appropriate technology prior to implementing the Communication Layer component are divided into generic parameters and specific parameters.

4.2.1.1 Generic Parameters

This section provides the definition of the generic parameters to be evaluated prior to selecting the technology for the Communication Layer. These are:

- **Maturity and Stability:** Stable and mature solutions are preferred.
- **Regularly Updated:** Technologies are preferred which are regularly updated.
- **Technical Up-to-Datedness / Appeal:** Technical up-to-datedness allows for using cutting-edge solutions and modern approaches for the INTUITEL development.

- **Non-Infecting:** As specified in the CA, solutions that come with an infecting license such as GPL should be avoided and non-infecting licenses should be preferred.
- **Code-Quality:** A good code quality is required for any technology that is selected as a base for components.
- **Extensibility:** A technology with well-defined extendibility may be considered, e.g. in terms of plugin mechanisms.
- **Community:** For clarifying questions and discussing possible problems of a technology, a strong community should be available.
- **Performance / Scalability:** Performance is an important criterion for all core components of INTUITEL although might differ from component to component.
- **Reuse of existing developments:** Reusing existing solutions of the partners should be preferred as those solutions will most likely be easier to adopt.
- **Platform (Portability):** Technologies should be preferred that allow a deployment into different platforms.
- **Open Standards Compliance:** Whenever an open standard exists, solutions should be preferred as long as those standards are openly accessible and applicable.
- **Interoperability (easy integration for all platforms):** If possible, technical solutions should be preferred that provides a good base for achieving interoperability.
- **Cost:** If possible, the technical solutions will be based on open source frameworks.

4.2.1.2 Specific Parameters

This section provides the definition of the specific parameters to be evaluated prior to selecting the technology for the Communication Layer. These are:

- **Reliable Messaging:** As INTUITEL deals with course data and progress, it must be ensured that the information (in the form of messages) transferred by the Communication Layer actually reach their destination.
- **Provide binary message exchange:** Files may have to be exchanged eventually, and when Communication Layer handles all messages between components, it should also be able to handle file exchanges.
- **Secure communication protocol:** The LMSs' users should have confidence that their data is securely transmitted. Therefore, the solution should include the possibility to have encrypted communication.
- **Signed messages:** In many cases, it is important to be able to guarantee that a message's origin is what it seems to be, especially focusing on the LO recommendations triggered by INTUITEL.
- **Provide message buffering:** When internet connection is not available meaning that the user is not logged into his LMS platform, messages must be buffered and sent as soon as possible.
- **Point-to-Point Messaging for component instances:** The main functionality of Communication Layer is to connect different components that might reside on different servers all over the Internet. In addition, as a cloud-based architecture is used in INTUITEL,

several instances of one component might need to send messages to the same components, e.g. the same USE proxy can send messages to different instances of LMSs.

- **Multi-recipient-messaging:** Some messages might need to be broadcasted or might need to target multiple components. Thus, the messaging load might be reduced, as the message would only need to be transferred to the message servers once and can be sent to the receivers from there.
- **Multi-instance-Server for cloud hosting:** As the INTUITEL architecture is planned to be a cross service for LMS, it has to ensure reliability, thus the Communication Layer solution should be able to work with multiple servers connected in a federation way, otherwise the whole distributed architecture would rely on one server running.
- **Provide configuration UI for routing:** As UIs tend to take a lot of implementation time, having a graphical configuration UI as part of the underlying technology stack would be a bonus.
- **Message partner presence awareness:** If the technology allows knowing which components or users are accessible, message load can be kept down, as messages could be sent only when the receiver is accessible.
- **Uses UTF-8:** The usage of UTF-8 will allow avoiding conversions between character sets.
- **Lightweight infrastructure:** As the messaging might be integrated in many LMSs, having a lightweight system that is easily set up and has minimum requirements is necessary.
- **Communication partner registry included:** Communication Layer will provide a way to show a registry of all component instances that are registered. If this were supported by the underlying technology, this would be a bonus.

4.2.1.3 Summary table

The following table summarizes the different parameters for the evaluation and their importance for a better selection.

	Parameter	Importance (- - - +/- + + +)
Generic Parameters	Maturity and Stability	+++
	Regularly Updated	+
	Technical Up-to-Datedness / Appeal	++
	Non-Infecting	++
	Code-Quality	+
	Extensibility	++
	Community	+
	Performance / Scalability	++
	Reuse of existing developments	++
	Platform (Portability)	+++
	Open Standards Compliance	+
	Interoperability (easy integration for all platforms)	+++
	Cost	+
Specific Parameters	Reliable Messaging	+++
	Provide binary message exchange	+
	Secure communication protocol	+++
	Signed messages	+/-
	Provide message buffering	+++
	Point-to-Point Messaging for Component Instances	+++
	Multi-recipient-messaging	++
	Multi-instance-Server for cloud hosting	+
	Provide configuration UI for routing	+
	Message Partner Presence Awareness	+
	Uses UFT-8	+++
	Lightweight Infrastructure	++
	Communication Partner Registry included	+

Table 6: Parameters of the selection criteria for evaluating the technologies

4.2.2 Possible Technologies

In order to realize the Communication Layer component several options have to be considered as base technology. The assessment of these technologies is presented in the following sections. Additionally a message format has to be defined or adopted from the chosen technology to submit metadata needed by INTUITEL in the transmitted messages.

Please note: The selection of possible base technologies in this subsection is based on experiments that part of the consortium made with different technologies. The best individual recommendations have been investigated. Therefore, the following selection will compare only the four most promising technologies.

- **REST / SOAP (WSDL) Web Services** - Using basic REST or WSDL based Web services is a widely accepted and most standards-conformant pattern for implementing all kinds of APIs all over the Web. Within INTUITEL this will also be used in the USE/TUG/LORE proxies, as those need to interfere with the LMSs provided by partners. For the Communication Layer these standards don't help much, as the high quantity of exchanged messages demands for a constant connection and makes the long request times of HTTP a show stopper.
- **XMPP** - XMPP is the abbreviation for eXtensible Messaging and Presence Protocol and is a standard for XML-based messaging over a socket connection upheld by connected clients. It is a stable and mature protocol. Facebook, Google and Apple use XMPP for their users' peer-to-peer messaging, which is a good indicator about its stability, maturity, performance and scalability. The extensibility of XMPP is organized by the XMPP Standards Foundation, which collects processes and formalizes the extensions into standards[7]. This is for example shown by Google Talk, which routes peer-to-peer voice chats or big file transfers over XMPP, and by other projects that also transfer video streaming data and much more performance-critical data. Interoperability is also well supported, as there is a client library for XMPP integration for every popular programming language, and there are multiple Open Source XMPP servers to choose from. XMPP keeps up a direct socket connection to the XMPP server via so called "long-polling", which means the communication first "logs in", which takes a bit because of authentication and content negotiation, and then sends a request with a very long timeout and gets an answer as soon as there is something to transmit. After a very long time, it is necessary to send another keep-alive packet, so that the connection does not break. For the login, it is needed a user account set in the XMPP server. As soon as there's something to transmit, it works very fast, as the connection is already there, and with all kinds of HTTP requests (REST and SOAP both use these) it is needed to make one request per call, which makes REST and SOAP much slower when there are a lot of requests. The payload of the messages is always wrapped in XML format.
- **ActiveMQ** - Apache ActiveMQ is a messaging bus based on a similar socket connection like XMPP made for high throughput and with many client libraries available for different

[7] http://xmpp.org/xmpp-protocols/xmpp-extensions

languages and platforms. It supports REST, SOAP, JMS, XMPP and other technologies, and integrates Apache Camel to use Advanced Integration Patterns. Additionally it supports SSL and JAAS, which is beneficial for the concerns arisen from INTUITEL regarding to login and credential problems. It supports different ways of clustering the servers. For INTUITEL, ActiveMQ provides a too broad set of configurable functionality and the overhead of too many possibilities and configurations is perceived as an impeding factor for INTUITEL. Another problematic factor is the real-world usage of ActiveMQ. The projects references all lie within the Apache community. Real-world problems like using socket connections through a router Network Address Translation or a server firewall can be a problem.

- **TSB** - TIE SmartBridge, from INTUITEL Partner TIE, is a Business Integrated Platform and an integration solution born from the B2B world. It provides tools for integration of back-office solutions, by implementing different interoperability strategies. It is compound of a hub for transferring single document/messages among the apps and services connected to it. One of the main features of the TSB is the B2B message exchanging broker able to transport B2B messages (XML, EDI, Flatfile…) from a source point to its destination, performing the adequate message transformations and routing. TSB also provides a mechanism for disaster recovery. The main applicability within INTUITEL would be the usage of the messaging bus, which communicates the different services through messages. New services can be added by using the plug-in paradigm resulting in an increase of the flexibility and the scalability of TSB.

The following tables summarize the analysis performed:

Parameter		Importance	REST / SOAP WS	XMPP	ActiveMQ + Camel	TSB
Generic Parameters	Maturity and Stability	+++	+++	+++	+++	+++
	Regularly Updated	+	+++	+	++	+++
	Technical Up-to-Datedness / Appeal	++	++	+++	++	+++
	Infecting Open Source License	NO	NO	NO	NO	NO
	Code-Quality	+	N/A	N/A	N/A	+++
	Extensibility	++	-	+++	+++	+++
	Community	+	+++	++	+++	-
	Performance / Scalability	++	+	++	++	++
	Reuse of existing developments	++	+++	+	+	+++
	Platform (Portability)	+++	+++	+++	+++	+
	Open Standards Compliance	+	+++	+++	+++	+++
	Interoperability	+++	+++	+++	+++	+++
	Cost	+	+++	+++	+++	-

Table 7: Comparison of technologies for the CL: Generic Parameters

Parameter		Importance	REST / SOAP WS	XMPP	ActiveMQ + Camel	TSB
Specific Parameters	Reliable Messaging / Receipt Acknowledgement	+++	+++	+++	+++	+++
	Provide binary message exchange	+	+++	+++	+++	+++
	Secure communication protocol	+++	+++	+++	+++	+++
	Signed messages	+/-	-	++	++	+++
	Provide message buffering	+++	-	+++	+++	+++
	Point-to-Point Messaging for Component Instances	+++	+++	+++	+++	+++
	Multi-recipient-messaging	++	-	+++	+++	+++
	Multi-instance-Server for cloud hosting	+	+++	+++	-	-
	Provide configuration UI for routing	+	-	+++	-	+++
	Message Partner Presence Awareness	+	-	+++	-	-
	Uses UFT-8	+++	+++	+++	+++	+++
	Lightweight Infrastructure	++	+++	+++	-	-
	Communication Partner Registry included	+	-	+++	-	-

Table 8: Comparison of technologies for the CL: Specific Parameters

4.2.3 Technology Selection

HTTP based communication through Web Services, i.e. SOAP or REST, is appropriate for basic connectivity. This basic connectivity acts like a point-to-point connectivity where the services are exposed in each *executing* component. Internally, the basic scenario includes a simple message queue.

For the "Advanced" scenario as defined in section 1.4, a more complex configuration of the Communication Layer shall be used. Under this statement and looking at the alternatives analysed in the previous section, there are not many differences between ActiveMQ and XMPP. The complexity of ActiveMQ is higher than that of XMPP, but both fit the INTUITEL problem space well. However, ActiveMQ is a more complex concept and it fails for the problem space requiring much more configuration rather than XMPP. Another drawback lies on the fact that most of the necessary concepts are not provided off the shelf, meaning that further developments may be required for the final implementation of ActiveMQ . This makes the selection of ActiveMQ not ideal for INTUITEL.

Between the last contenders - XMPP and the routing of TSB, the differences are not that big. However, TSB can be seen as a more complete tool which adds to the Communication Layer other services like message transformation, workflow editors and message queuing. The advanced CL therefore will be realized as a hybrid solution XMPP-TSB. The main advantage it has relies on the implementation side: Each INTUITEL component will use the same calls as before, internals are completely hidden from the components through the usage of another client library. With this one, the INTUITEL components will interact with the XMPP server (which is in charge of routing the messages to the TSB), while the INTUITEL backend will implement the TSB as it will access the services offered by INTUITEL, like e.g. the Engine. Additionally, real world applications using XMPP, including the messaging systems of Google, Apple and Facebook, show that XMPP is a scalable and mature first class technology, with a good stability and performance. Interoperability is a given for XMPP as there is some form of client library for every major programming language, and there are multiple XMPP servers to choose from[8].

Both XMPP and TSB transfer XML data. In this, binary data can be easily included in serialized form. The metadata that cannot be delivered through the basic XMPP message structure can be added as XML metadata. The structure of the message body is specified by the receiving components, as they need to define which data is needed in the XML message body in order to carry out tasks or respond appropriately.

Of the additional services offered by TSB, the transformation and the workflow editor may be relevant for future extensions of INTUITEL. Within the transformation, the content of a message can be translated from one format to another one. Within the workflow editor it is possible to create personalized routes or actions for the messages exchanged, e.g. the transformation would require a workflow to route the message from the source to the transformation engine, transform and then send the message from the transformation engine to the destination. In any case, these services are not a requirement for the current version of INTUITEL.

[8] http://xmpp.org/xmpp-software