Deliverable 1.2

USE/TUG/LORE Interface

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)	

SEVENTH FRAMEWORK PROGRAMME

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.:	D1.1
Document name:	D1. USE/TUG/LORE Interface
Work Package	1
Associated Task	1.2, 1.3, 1.4, 1.5, 1.6
Nature [1]	P
Dissemination Level [2]	RE
Version:	1.0
Actual Submission Date:	2013-09-30
Contractual Submission Date	2013-09-30
Editor: Institution: E-mail:	Elisabetta Parodi Lattanzio Learning SpA elisabetta.parodi@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
0.01	2013-06-05	Document drafted	Elisabetta Parodi	ELS
0.02	2013-07-17	Draft revised	Elisabetta Parodi, Agostino Fanti	ELS
0.03	2013-09-13	Merged contributions about implementation in Moodle	Elena Verdú Pérez, Juan Pablo de Castro, María Jesús Verdú	UVA
0.04	2013-09-19	Added contributions about: - implementation in Crayons - implementation in eXact (basic version, to be revised)	Ehm Kannegieser, Daniel Szentes Elisabetta Parodi, Enrico Bruzzone	IOSB ELS
0.05	2013-09-20	Refinement of ELS contribution Added contributions about implementation in ILIAS and verification in ILIAS	Elisabetta Parodi Kevin Fuchs, Steffen Burger, Peter Henning Luis de la Fuente	ELS HSKA UNIR
0.06	2013-09-25	Finalization of ELS contribution Integration about Geolocation and testing for Moodle Added contributions about implementation in Clix Small refinements to Moodle section	Elisabetta Parodi, Enrico Bruzzone Elena Verdú Pérez Sven Steudter, Uta Schwertel Elena Verdú Pérez	ELS UVA IMC UVA
0.07	2013-09-26	Added paragraph about tests results in ILIAS Added section about eXact LMS GUI and conclusions	Luis de la Fuente Valentín Elisabetta Parodi, Enrico Bruzzone	UNIR ELS

1.0	2013-09-30	Final refined version	Peter Henning Elisabetta Parodi	HSKA ELS

Distribution List

Date	Issue	Group
2013-07-17	Request for comments & contributions	WP01 mailing list
2013-09-13	Request for comments & contributions	WP01 mailing list
2013-09-19	Request for comments & contributions	WP01 mailing list
2013-09-25	Request for comments & contributions	WP01 mailing list
2013-09-27	Request for comments & contributions	WP01 mailing list

List of Contributions

Date	Organization(s)	Person(s)	Contribution
2013-09-11	UVA	Elena Verdú Pérez, Juan Pablo de Castro, María Jesús Verdú	Documentation about implementation in Moodle
2013-09-17	IOSB	Ehm Kannegieser, Daniel Szentes	Documentation about implementation in Crayons
2013-09-17	ELS	Enrico Bruzzone	Draft documentation about implementation in eXact
2013-09-17	HSKA	Kevin Fuchs, Steffen Burger, Peter Henning	Documentation about implementation in ILIAS
2013-09-17	UNIR	Luis de la Fuente	Documentation about verification of implementation in ILIAS
2013-09-23	ELS	Enrico Bruzzone, Elisabetta Parodi	Updated screenshots, added general remarks
2013-09-23	UVA	Elena Verdú Pérez	Integration about Geolocation and testing for Moodle
2013-09-23	IMC	Sven Steudter, Uta Schwertel	Documentation about implementation in Clix
2013-09-24	UVA	Elena Verdú Pérez	Section 3.2.1: explanation of a new file rest.php; Section 3.2.2: some lines deleted as those parameters for endpoints are not more needed.
2013-09-25	UNIR	Luis de la Fuente Valentín	Paragraph about tests results in ILIAS

Table of Contents

List of figures

List of tables

Codelisting index

1 Glossary

Term	Explanation
Course	The topmost cognitive container of an INTUITEL enhanced system, equivalent to the Knowledge Domain (KD).
INTUITEL-MUST	A feature type for INTUITEL components. Components carrying this feature type must be present to allow INTUITEL functionality and cannot be mimicked or emulated by other means.
INTUITEL-SHOULD	A feature type for INTUITEL components. Components carrying this feature type are needed for INTUITEL functionality, but may be mimicked or emulated by other INTUITEL components. Example: The USE interface is needed for INTUITEL functionality, but it may be impossible for technical reasons of a particular LMS to provide this. In this particular case, one may emulate the USE interface by asking questions via the TUG interface.
INTUITEL-MAY	A feature type for INTUITEL components. Components carrying this feature type are needed for INTUITEL functionality, but may be mimicked or emulated by other INTUITEL components. Example: The USE interface is needed for INTUITEL functionality, but it may be impossible for technical reasons of a particular LMS to provide this. In this particular case, one may emulate the USE interface by asking questions via the TUG interface.
Knowledge Domain (KD)	The topmost cognitive container of an INTUITEL enhanced system, equivalent to a course. The KD element encapsulates a set of Concept Containers to create an outline for a domain of knowledge. Such CCs are e.g. thematically related collections of pages (learning modules), tests, surveys, files, media objects.
Learner	A learning person using the INTUITEL enhanced LMS.
LMS	Learning Management System
LPM	Learner Progress Model
LORE	Learning Object Recommender, an interface provided by the LMS
Profile of a LMS	A set of attribute-value pairs describing the capabilities of the LMS.
REST	Representational State Transfer
SCORM	Sharable Content Object Reference Model, a format combining metadata and learning content.
SLOM	Semantic Learning Object Model, a format combining metadata and learning content for the INTUITEL system. Note, that SCORM allows for extensive metadata content – it is therefore possible to store even the extended information needed for INTUITEL in SCORM courses – where is then ignored by the LMS.
TUG	Tutorial Guidance, an interface provided by the LMS
USE	User Score Extraction, an interface provided by the LMS
UUID	Universal Unique Identifier, a 16 octet (128 bit) data value uniquely identifying system messages. Specified according to the Open Group UUID standard, therefore also containing a time stamp[3]. Example: 12345678-1234-abcd-ef12-123456789012

[3] See http://pubs.opengroup.org/onlinepubs/9629399/apdxa.htm on how the timestamp is contained in this UUID

2 Introduction

2.1 INTUITEL & LMSs

The objective of INTUITEL is to enhance state-of-the-art e-learning content and Learning Management Systems (LMS) with features that so far have been provided only by human tutors.

A Learning Management System (LMS) is a software application for the administration, documentation, tracking, reporting and delivery of e-learning education courses or training programs. There is no standard industry definition or published standard defining the components of an LMS, but several features are common: creation of class rosters, control over registration processes, and the ability to create waiting lists; upload and management of documents containing curricular content; delivery of course content over web-based interfaces, most often allowing remote participation by the instructor or pupil; creation and publication of course calendars; interaction between and among students, such as instant messaging, email, and discussion forums; methods of assessment and testing (like creating pop quizzes).

LMSs range from systems for managing training and educational records to software for distributing online or blended/hybrid college courses over the Internet with features for online collaboration. Colleges and universities use LMSs to deliver online courses and augment on-campus courses. Corporate training departments use LMSs to deliver online training, as well as automate record-keeping and employee registration.

An INTUITEL-enabled system constitutes an integrated learning environment that configures itself in response to any learner, monitors his/her progress and behaviour, combines these data with pedagogical and methodological knowledge and then by automated reasoning deduces optimal guidance and feedback. The deductive process may include the current learner performance, the daily learning attitude and emotional setting of the learner, personal aspects like gender, culture and age as well as environmental aspects like available communication bandwidth, ambient noise level, screen size and type of access device. INTUITEL therefore will be a step towards a global learning cloud, where personalized technology-enhanced learning is available for any person at any place, with any access device and under any external condition, including mobile learning scenarios.

The INTUITEL-enabled LMS then plays the role of a pedagogically skilled teacher, transparently guiding the learner towards the required competencies. For realizing the first prototypical implementations of INTUITEL-enhanced LMSs, five systems have been selected and related providers/developers have been added to the consortium, namely:

- Two open source LMSs, ILIAS and Moodle; related extensions are carried on by HSKA and UVA partners
- Three commercial LMSs, namely Crayons, Clix and learn eXact, whose extensions are developed by related providers, respectively IOSB, IMC and ELS.

In order to integrate the INTUITEL core reasoning components and the LMSs, communication interfaces have been agreed and LMSs extensions developed. The present document describes the implementation and verification of such implementations.

2.2 USE/TUG/LORE Data Model – Recall

As said before, to make possible the integration between INTUITEL core components and an LMS, a so-called USE/TUG/LORE interface was agreed and related extensions have been developed by LMSs providers and developers.

The TUG interface carries out a dialog between the learner and the INTUITEL system. A TUG Interface *must* be present in any INTUITEL enhanced system (feature type INTUITEL-MUST).

The LORE interface makes a recommendation by the INTUITEL system known to the learner. It is, in its simplest implementation, a special case of the TUG interface and therefore present in any INTUITEL enhanced system (feature type INTUITEL-MUST). However, since more complicated ways exist to present the recommendation, it is treated separately.

The USE interface extracts data about the learner and his environment from the LMS, therefore it *should* be present in INTUITEL enhanced systems (feature type INTUITEL-SHOULD). It is, with limited functionality, also provided by the TUG interface, which therefore may serve as an emulation of USE.

An extended description of USE/TUG/LORE interface is presented by Deliverable D1.1 XML Schema for USE/TUG/LORE.

2.3 Executive Summary: Structure of the current document

This document describes the extensions developed for the five target LMSs in order to make them able to communicate with the INTUITEL core reasoning components. Therefore the implementation section is divided into five parts, one for each LMS. Each part presents an overview of the solution, a short description of the "work behind the scenes" for providing the given interfaces, the verification of the developed implementation and additional notes and comments.

It has to be noted, that the software developed in Tasks 1.2 – 1.6 and documented here is not available as standalone software components that may be distributed to a reviewer for a simple test. Rather, the USE/TUG/LORE interfaces are present in special installations of learning management systems, any test therefore consists of sending special REST messages to one of the LMS and checking for the appropriate return as defined in Deliverable D1.1.

- For the two Open Source LMS, these REST endpoints are found at
  - http://ice-karlsruhe.de/intuitel-ilias
  - http://itastdevserver.tel.uva.es/moodle2/blocks/intuitel/rest.php

and are hereby open for a review. However, for convenience of the reviewer the consortium wishes to point out that under the URL

- http://tel.unir.net/intuitel-tests/
- User: intuitel, Password: showmethetests

an automatic test suite is available (created by INTUITEL partner UNIR independently from the developing partners HSKA and UVA) which checks these two INTUITEL installations automatically at each midnight against the data model from deliverable D1.1. Note, that these tests are described in section 3.1.10 of the present paper and that their results are open for the review process. At the time of submission of the present paper, the majority of these test gives an expected result and 2-3 (out of 75) show less than sufficient results.

- For the three commercial LMS Clix, eXact and Crayons, such an automatic testing is currently not available for licensing and security issues. However, test cases carried out manually and screen shots have been documented here, consequently the proper verification has been carried out. The consortium expects that live hands-on testing will be available at the scheduled review meeting for INTUITEL (Dec. 10, 2013).

Note in this context also, that the Tasks 1.2 – 1.6 are still running until the end of November, 2013. **The project milestone M2, e.g. the availability of the USE/TUG/LORE interfaces therefore is considered to be achieved as planned.**

3 Implementation

3.1 USE/TUG/LORE in ILIAS (HSKA)

The following describes how INTUITEL is integrated into the ILIAS system. The design follows an approach that requires only a minimum of changes in the ILIAS code implying only a loose coupling of INTUITEL and the ILIAS system.

The design presented in this document starts on a high and abstract level describing roughly the modules of the integration component in respect of the message exchange between them. In subsequent steps this model is refined into software components and the according interfaces. Finally the package model forms the last step of refinement, yielding the concrete structure of implementation.

The functionality of the integration component is reduced to a minimum. In the best case it acts only as an intermediate component delegating INTUITEL actions to the LMS. The integration component is implemented in Java being run by the application server Tomcat in combination with the Apache Webserver.

Some parts of the integration component need to read data from the ILIAS database. As the database scheme of ILIAS may change between different ILIAS versions the software parts being

responsible for the database connection have to be easily exchangeable. The functionality related to database connections is therefore encapsulated in a separate software component.

Thus, some analysis of the ILIAS database is necessary to determine those pieces of information that – on the one hand – are already contained by the database and – on the other hand – those that have to be added to the database. The latter case requires modifications of both the database and some PHP files of the ILIAS system.

In order to give the user's browser the capability to send and receive USE/TUG/LORE messages, the PHP- and HTML-files of the ILIAS system are extended with JavaScripts to send AJAX requests. To display the LORE/TUG messages in the browser also JavaScript is used. These scripts use DOM access to create new HTML objects within existing ones. This ensures that the original PHP- and HTML-files of ILIAS are only marginally changed. Indeed, no modification of the ILIAS source code is necessary as far as the presentation of TUG and LORE messages to the user is concerned.

3.1.1 Architecture

3.1.1.1 Software Layers, Modules and Messages

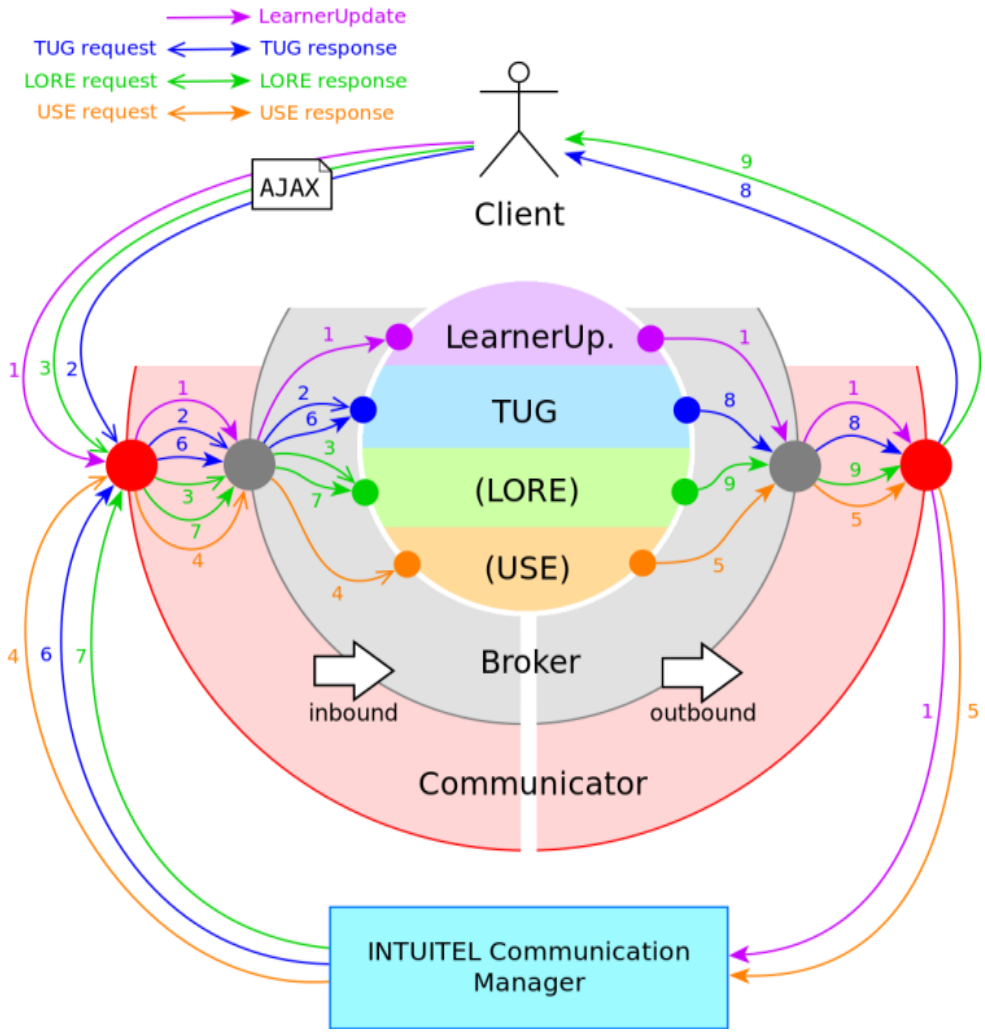

Figure 1: Software layers, modules and messages

The integration component connecting the LMS with INTUITEL is implemented with Java consisting of three layers as shown in Figure 1. The Broker and the Communicator are each divided into two sub-modules representing the inbound and the outbound direction of messages coming from or being sent to the INTUITEL Communication Manager. This architecture is necessary to avoid double coupling of modules for the purpose of maintainability. The integration component comprises three layers: [4]

[4] Note, that within this document an ILIAS specific naming convention is used, e.g. "Communicator" refers to e special part of the ILIAS INTUITEL implementation and is not to be confused with the INTUITEL communication manager. This naming convention has been introduced for a clear view on the abstract software design.

- The *Communicator* which is in charge of sending and receiving messages from and to the INTUITEL Communication Manager.
- The *Broker*, that passes messages between the *Communicator* and the *Core*.
- The *Core* finally contains the modules that handle LearnerUpdates, TUG, LORE and USE messages. There are even more messages, each corresponding to an according Core module. However, due to clarity, only the four mentioned modules are illustrated in Figure 1. Some Core modules finally are connected to the ILIAS database. The database is encapsulated by a database connector module, the interface of which is used by these Core modules.

These three layers are now explained in more detail.

Communicator

The Communicator offers a communication point for the INTUITEL Communication Manager and the user's browser. This communication is indicated by two red circles – one for inbound and the other one for outbound messages. As mentioned earlier, when refining this model to software components the inbound and outbound parts will be separated from each other to avoid double coupling of components.

The Communicator will provide a simple REST service to accept messages. However, the Communicator could also provide any other communication method.

The Communicator is absolutely unaware of the content, sources and destinations of the messages it processes. Its only task is the provision of the REST interface as well as passing and receiving messages from and to the layer on the next level which is the Broker described below.

Broker

The Broker represents the second layer of the design. Again – it comes with interfaces that connect it to the Communicator and the Core – indicated by the coloured circles in Figure 1. Similar to the Communicator, there is an inbound and an outbound direction.

The Broker's task on the inbound side is to receive messages from the Communicator and delegate them to the appropriate Core module which can be the TUG, LORE, USE, LearnerUpdate module and other modules respectively, which are described below. This means that – in contrast to the Communicator – the Broker knows about the destinations and sources of messages in order to determine the fitting Core module to which it delegates the message. However, the Broker is indifferent to the contents of messages.

On the outbound side the Broker takes messages from the Core components and delegates them to the Communicator.

Core

The Core contains the modules that are in charge of handling TUG, LORE, USE messages and LearnerUpdates as well as other INTUITEL services. On the outbound side, each of them receives corresponding messages from the Broker. They read the message payloads and take appropriate action. On the outbound side they delegate TUG, LORE, USE and LearnerUpdate messages to the Broker which in turn passes them to the Communicator. The Communicator finally passes the messages either to the INTUITEL Communication Manager or to the user's browser.

3.1.1.2 General Services

The Core contains modules that provide general services for the TUG, LORE, USE and LearnerUpdate modules as well as for other Core modules. This is illustrated by Figure 2. Such general services are:

- Marshalling services to transform XML payloads for processing
- Database connectivity to retrieve data from the ILIAS database
- Error handling
- Logging functions

Encapsulating these methods in separate services contributes to the ability to replace and modify the according functions without affecting other modules. One could – for example – use other data formatting than XML or migrate to another database. A quite realistic scenario is the one where changes of the database scheme of the ILIAS database happen, i.e. due to migrations to newer versions. Hence, placing such functionality into separate components contributes to the maintainability and extensibility of the entire system. The single services are described in more detail below.

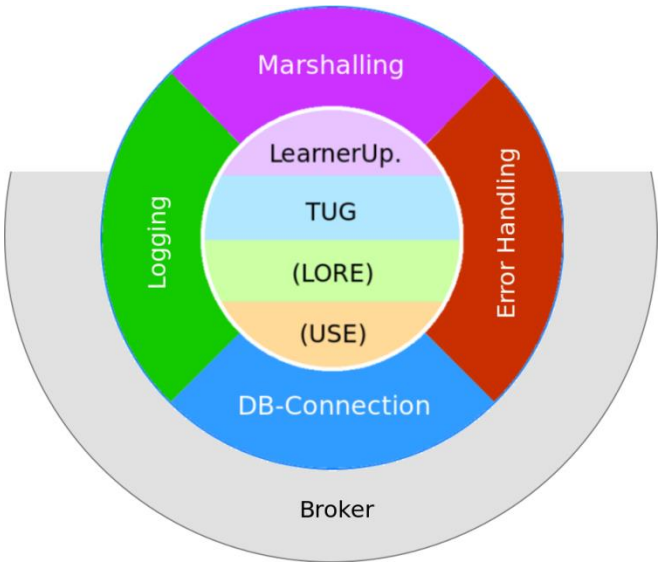

Figure 2: Core modules and general services

Marshalling

The marshalling service is used by all Core modules that have to handle INTUITEL messages. Both the Communicator and the Broker are unaware of message contents being only responsible for the delivery and delegation of messages to the next layer.

The Broker only knows about the type of messages (TUG, LORE, LearnerUpdate, USE etc.) whereas the Communicator has no knowledge about this at all. For that reason, the Core modules must parse the XML bodies of those messages. Since this is a task that is quite similar within every Core module, it is worth outsourcing it to a separate, partly generic service that can be used by all Core modules. In this regard, marshalling means transforming XML contents into Java objects that can be consumed by the Core modules.

Error Handling

Since error handling comprises procedures that are similar in every Core module, it is outsourced to a separate module which in addition provides debugging functionality.

Logging

Similar to Error Handling and for the same reasons, logging functionality is outsourced to a separate logging service.

DB-Connection

The LearnerUpdate module and USE module have to access the ILIAS database. The first one needs to resolve session ids to user ids, which is contained by the database. The latter one needs to retrieve environmental and performance data in order to enable the INTUITEL reasoner to generate recommendations for the user.

In order to decouple database connectivity from the integration component and make it thereby maintainable and exchangeable, database access is encapsulated by the DB-Connection service. Database access is provided to the Core modules by a cohesive interface. This necessitates from the fact that the ILIAS system is permanently evolved by the ILIAS developer team which results in frequent updates. These updates also involve modifications of the database. Encapsulating database connections within a separate service allows adapting it to such changes without the need of modifying the other modules.

Remarks

It is important to mention that any communication – no matter from which source or to which destination, inbound or outbound – must pass through the Communicator. There is no other way to communicate with modules of the other layers. This guarantees that the single layers can be replaced or modified without affecting the other layers – as long as they have clearly defined, cohesive interfaces.

For example, the inner logic of the Communicator could first be developed in a straight forward manner and later be replaced by components that are provided by other developers. The Core module – as another example – may lack the LORE or USE module or even both of them. In this case, the missing modules have to be emulated by the TUG module. Restricting communication in the mentioned way makes it possible to remove or add a USE or LORE module without the need of modifying the Broker and the Communicator. To make this approach work a careful interface design is necessary which is subject of section 3.1.9.5.

In fact, all parts of the integration component except for the Core modules, do not contain any LMS specific logic. Even the definition of the Core modules' interfaces that connect the Broker with the Core modules can be done in a LMS-independent style. This way only the inner logic of the Core modules has to be adapted to meet the requirements of a particular LMS.

3.1.1.3 Message Sequences

As sketched in the introduction, TUG and LORE dialogues are embedded into the client's browser using JavaScript and AJAX functionality. Based on this assumption, the message sequences are due to the push scenario which is explained in Deliverable 1.1. The following explanations refer to the message sequence chart from Deliverable 1.1 which is shown in Figure 3 below. In Figure 1 above, the coloured and numbered message paths correspond to messages in the message sequence chart in Figure 3.

Messages Step by Step

1. At the beginning, the user opens a page in the LMS, causing an AJAX script to send a LearnerUpdate message to the Communicator of the LMS (path 1 in Figure 1). The LearnerUpdate first encounters the Broker which delegates the message to the LearnerUpdate module. The latter one passes the message to the Broker which in turn passes it to the Communicator. Finally the LearnerUpdate message is sent to the INTUITEL Communication Manager. This makes the INTUITEL system trigger the reasoning process.
2. In the meantime, the user's browser again calls AJAX scripts to send a TUG and a LORE request to the Communicator (path 2 and 3 in Figure 1). Again the message passes the Broker which sends it to the TUG and LORE module, respectively.

The INTUITEL system is still busy preparing the reasoning process. Therefore, the TUG/LORE modules must store the client's TUG/LORE requests as long as there is no appropriate response message.

3. In the meantime, the INTUITEL system emits a USE request in the context of the preparation of the reasoning process. The USE request is sent to the Communicator (path 4 in Figure 1). Exactly as the messages described before, the USE request passes the Communicator and the Broker until it is delegated to the USE module.

   The USE module now has to retrieve the information that is needed to satisfy the request. In the case of ILIAS the USE module gathers this data from the ILIAS database. It has therefore to be connected to the ILIAS database. As described earlier, the database connectivity will be provided by a separate module within the Core.

4. When the USE module has gathered all necessary data it sends a USE response to the INTUITEL Communication Manager passing first the Broker and second the Communicator (path 5 in Figure 1).
5. As the INTUITEL reasoner now has all the information it needs for the reasoning process the reasoner begins its work. After it has finished a LORE and a TUG response are sent to the Communicator. These two messages correspond to the previously sent TUG/LORE requests which the TUG/LORE modules are still expecting to be satisfied (path 6 and 7 in Figure 1).
6. Passing the Communicator and the Broker, the messages are matched by the TUG/LORE modules to the former requests they have still stored. As a result the TUG/LORE modules generate the corresponding responses that are sent to the browser where JavaScript and AJAX functions display the responses to the user (path 8 and 9 in Figure 1).

In the subsequent sections we give a closer description of INTUITEL messages and their representations as modules within the integration component respectively.

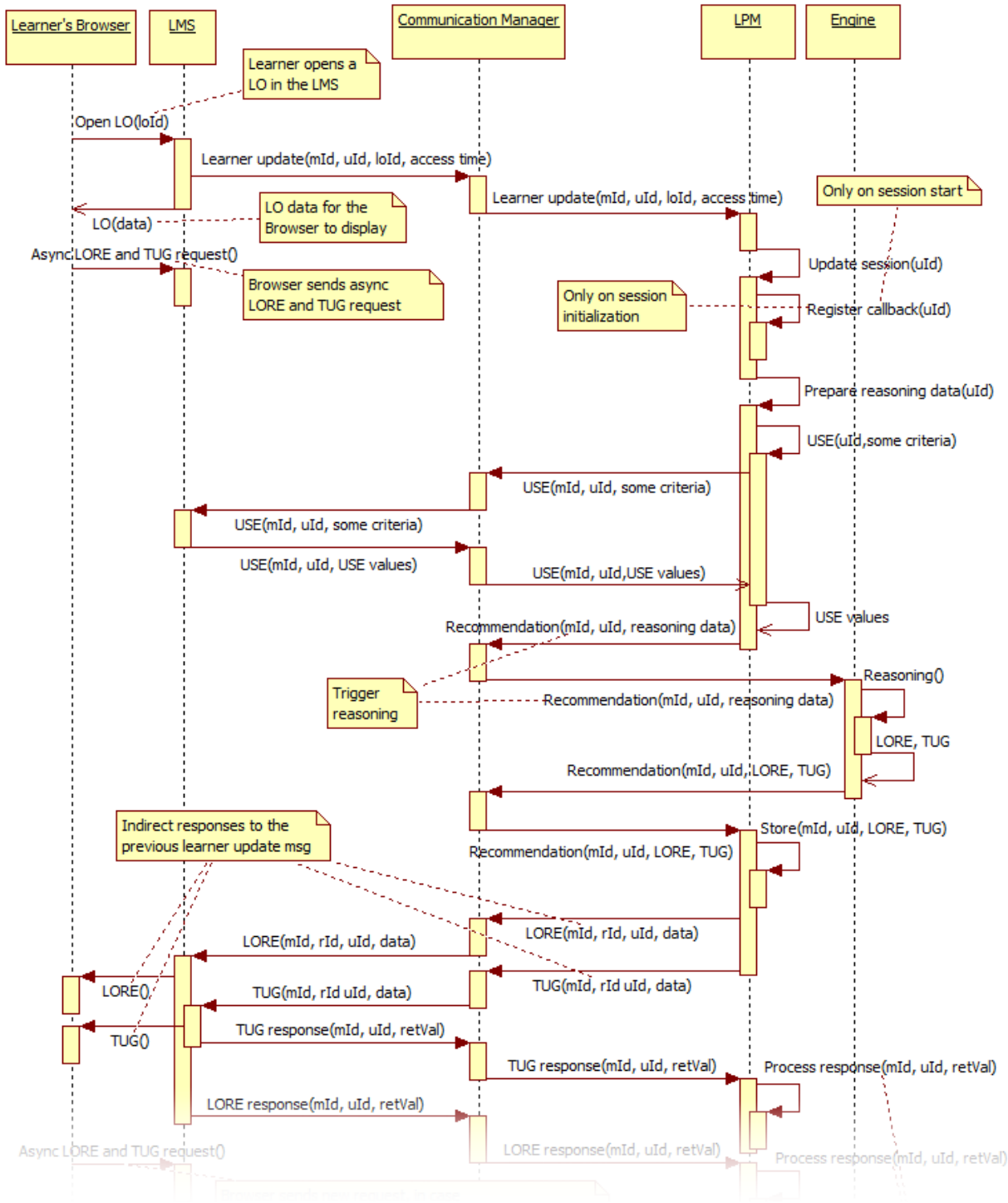

Figure 3: Message sequence chart for the push scenario

3.1.2 LmsProfile

The LmsProfile is not shown in Figure 1. Just like the TUG/LORE/USE and LearnerUpdate modules it belongs to the Core. It provides basic information about the LMS including LMS type, LORE level and so forth. The implementation of the LmsProfile module is straightforward. In the easiest case the information is provided in a static hard-coded way. Additionally data can be retrieved from the ILIAS database using the DB-connector module. The according message is then delegated to the Broker.

3.1.3 LearnerUpdate

The LearnerUpdate is the most simple *Core* module. It only receives LearnerUpdate messages from the learner's browser and delegates them to the INTUITEL Communication Layer by the use of the Broker's interface.

3.1.4 LO Mapping and Inventory

The LoMapping module also is implemented in a straightforward manner. Given a set of object ids and search patterns respectively, the LoMapping module delegates these requests to the DB-connector module and generates an according response message.

3.1.5 Authentication

Authentication is not necessary for ordinary ILIAS users. It is only designed for INTUITEL editors in order to identify those LOs for which the respective user has write access. For authentication an appropriate user id is required matching the respective user id in the ILIAS system. The Authentication module initiates a database lookup for the given user id and the learning objects this user has write access to. Again the request is delegated to the DB-Connector module.

3.1.6 TUG

The TUG module performs the following tasks:

- Receiving TUG requests from the user’s browser
- Receiving TUG responses from the INTUITEL Communication Manager whereby the TUG responses are the INTUITEL system's reaction to the client's LearnerUpdate message.
- Sending TUG responses back to the browser.

3.1.7 LORE

The LORE module works similar to the TUG module and is in charge of the following tasks:

- Receiving LORE requests from the client
- Receiving LORE responses from the INTUITEL Communication Layer (the LORE responses are the result of LearnerUpdate messages)
- Sending LORE responses to the user's browser which requested this LORE response

The LORE module may be missing, so that the TUG module will have to emulate the LORE interface.

Just like the TUG module, it is up to the LMS how the presentation of the TUG messages to the user is implemented.

The here described scenario of message exchanges between the client and the integration component is the favoured way as far as loose coupling between the client and the integration component is concerned. However, the disadvantage of this arises from the lack of detailed information that cannot be accessed from within the client. For this reason we must extend some messages that are defined in Deliverable 1.1.

Note that such an extension does not violate the data model in Deliverable 1.1 since this affects only the communication between the browser and the integration component. Other components of INTUITEL are not concerned by this. As sketched earlier, for the communication between the browser and the integration component, one could in principle use a completely self-designed communication with a completely different data model. Nevertheless, we decide to build on the data model that is used by the rest of the INTUITEL system since this creates a clear and cohesive design.

The client only offers session ids and learning object ids. But it does neither provide the name nor the description of a learning object. Actually, LORE messages which contain lists of learning objects that are recommended to the learners only include the ids and the priority levels of learning objects as defined in Deliverable 1.1. Therefore, the client would have no information about the learning objects to display for the learner.

The only way to solve this problem, is to extend the LORE message data structure such that it contains the name and the description of the respective learning object. The extended data model for LORE messages between the client and the integration component is listed below. The underlined parts indicate the elements added to the original LORE data model.

```
<INTUITEL>
   <Lore uId="some user ID" mId="some uuid" [rId="some uuid"]>
      <LorePrio loId="some LO id" value="some value between 1 and 100"/>
         <LoreLoName>name of the lo</LoreLoName>
         <LoreLoDesc>description of the lo</LoreLoDesc>
      </LorePrio>
      <!-- Repeated if necessary -->
   </Lore>
   <!-- Repeated if necessary -->
</INTUITEL>
```

Codelisting 1: LORE message from INTUITEL

Both the name and the description must be added to this message by the LORE core module. The LORE module retrieves this information from the ILIAS database by the use of the DbConnection module. To visualize this, remember Figure 1 and Figure 2.

3.1.8 UsePerf and UseEnv

The USE module handles USE requests and sends appropriate USE responses back to the INTUITEL Communication Layer. It has to process two different types of USE requests: those reflecting performance-specific data (UsePerf) and such that refer to environmental data (UseEnv).

For this purpose, the USE module reads the according data from the ILIAS database. This approach assumes that the required data are present in the ILIAS database. Of course, for some data this prerequisite cannot be satisfied. Therefore, the ILIAS system has to be modified such that the missing data are written into the database.

Similar to the LORE module, the USE module may be missing. As a result the USE interface is emulated by the TUG module asking the required information directly from the user.

3.1.9 Notes about Implementation

3.1.9.1 TUG and LORE Messages in the ILIAS GUI

The following section describes how TUG and LORE messages are presented to the user and how this is realized in the ILIAS system. We only use JavaScript methods to inject code into existing PHP and HTML pages. Moreover, by using the skin mechanism of ILIAS the user can activate and deactivate INTUITEL messages.

ILIAS Skin Mechanism

In order to integrate the presentation of TUG and LORE messages into the ILIAS system, the skin mechanism coming with ILIAS is used. By this skin mechanism the user is allowed to select a certain layout and coloring scheme for the ILIAS web interface. The system administrator determines which default skin is presented to the ILIAS user, and whether it may be changed by the user at runtime.

A skin can be added to ILIAS by including individually adapted templates within the customizing area of ILIAS. If the skin selection is permitted, it may be done by users under their personal settings as shown in Figure 4. We implement one or more INTUITEL-enabled skins which the user may select. A user may also select a skin which is not INTUITEL-enabled and therefore does not provide INTUITEL functionality.

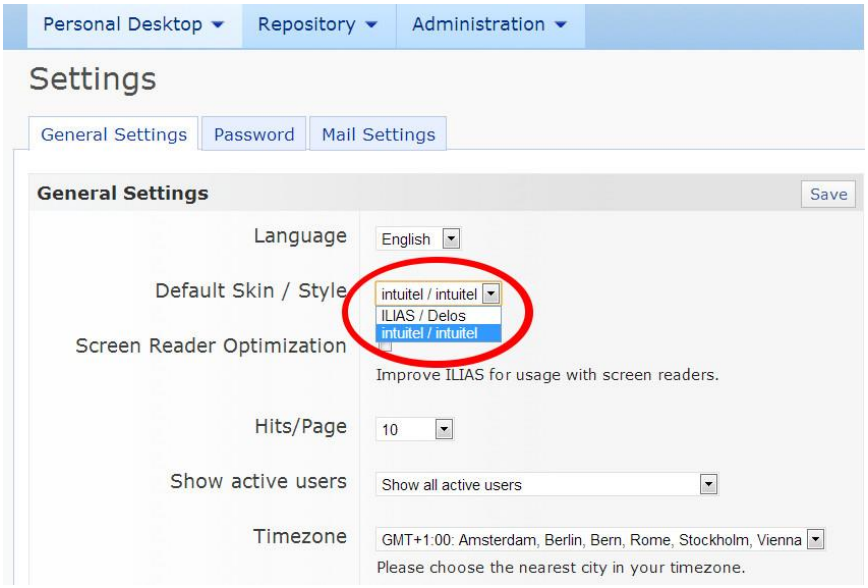

Figure 4: Skin selection in ILIAS

It is important to notice that the deactivation of the INTUITEL skin only affects the presentation of the messages while INTUITEL is still running in the background – invisible to the user. In specific this means that the LMS keeps on sending LearnerUpdates to and receiving TUG and LORE messages from the INTUITEL system. INTUITEL consequently still triggers the reasoning processes and sends appropriate responses to the LMS which, however, are ignored by the browser. Hence, the entire set of INTUITEL processes is still running in the background. The intention is that whenever the user decides to reactivate the INTUITEL skin, the INTUITEL reasoner can keep on generating recommendations based on the user's actions and transitions in the past – even if the INTUITEL skin was deactivated before.

Message Popups

When new TUG or LORE messages arise, the user faces a popup message in the upper right corner of the browser window as illustrated by Figure 5. The popup's position is fixed relative to the upper right corner of the browser window, so it is invariant in respect to scrolling positions. The user can either react on the message immediately or close the popup or put it aside as shown in Figure 6. When put aside, the popup window informs about new messages with a flashlight signal and optionally a sound signal that can be activated or deactivated by clicking a respective button at the right top of the popup window.

Figure 5: Message popup

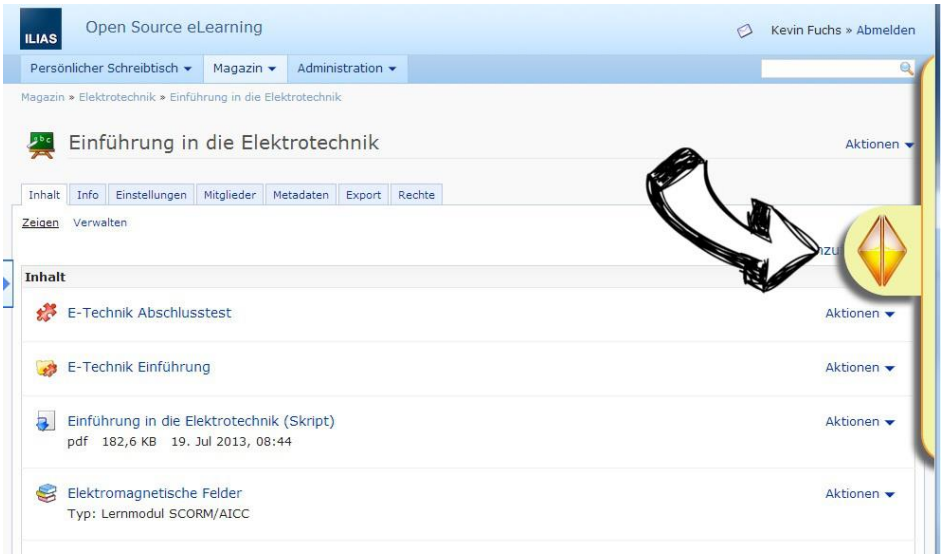

Figure 6: Hidden popup, flashlight signal

JavaScript and DOM

The JavaScript implementation uses the Document Object Model (DOM) to access the HTML output of the ILIAS web pages. Following this approach no manipulation of the original ILIAS files is necessary. The corresponding JavaScript functions simply access the body tag of the HTML page and insert the necessary code to render the message popup. Based on this, any desired kind of HTML code can be inserted – including forms or scripts providing interactivity.

This way, indeed, no changes to the ILIAS sources are needed as there is only a simple JavaScript that has to be included by the main template of ILIAS. As sketched previously, this takes place within the skin template of ILIAS. Moreover, this technique is rather generic and can also be applied to other web-based LMS.

Preserving Popup States

When including the JavaScript functions in the main template of ILIAS, a problem occurs when a learning module is spread over several pages through which the user navigates. Every time a new page is loaded the last message shown by the popup disappears. This can be very annoying when the user has put a message aside with the intention to react on it later. The message is then lost and cannot be recovered by the user.

To avoid this, the whole LMS web GUI is wrapped by a frame which is invisible to the user. The frameset includes JavaScript providing an object that saves the state of the popup window and the last Learning Object the user has visited. This embraces the following:

- The title as well as the content of the last message.
- Display status saying if the user has put the window aside
- The id of the last Learning Object the user has visited

Referring the first point, there may be more than one message that the user has not considered yet. Every time a new message arises, older messages are discarded. This makes sense since a TUG or LORE message always is a result of an INTUITEL-internal reasoning process that considers the entire path a user has taken through the learning material as yet. On that account, older messages become obsolete at the moment a new message is received.

If, for some reason, it is required to keep older messages, the JavaScript for rendering the popup window can be modified as follows: when a new message is received, the popup window is not replaced. Instead, a new window is created, covering the old one. When the user closes the popup window or puts it aside, the next older message appears. However, for now, this is not implemented but it is taken into consideration for the future.

Wrapping the entire ILIAS page in a top frameset may interfere with other ILIAS-internal framesets. For example, the repository explorer uses a frameset consisting of one frame presenting the repository browser and another one showing the content. At particular locations this frameset is removed when the user navigates to another page. This is done by targeting the correspondent links to the top window of the browser, which causes the removal of all frames, which also includes the INTUITEL top frame. However, within the ILIAS code, this can easily be avoided by replacing all link targets addressing the top window of the browser with link targets addressing the id of the INTUITEL top frame. This way, only those framesets being children of the INTUITEL frameset are removed but not the top frame itself. The replacement of the target links is also done by a JavaScript function. Therefore, the original ILIAS code remains unchanged.

3.1.9.2 Communication between Top Frame and Main Template

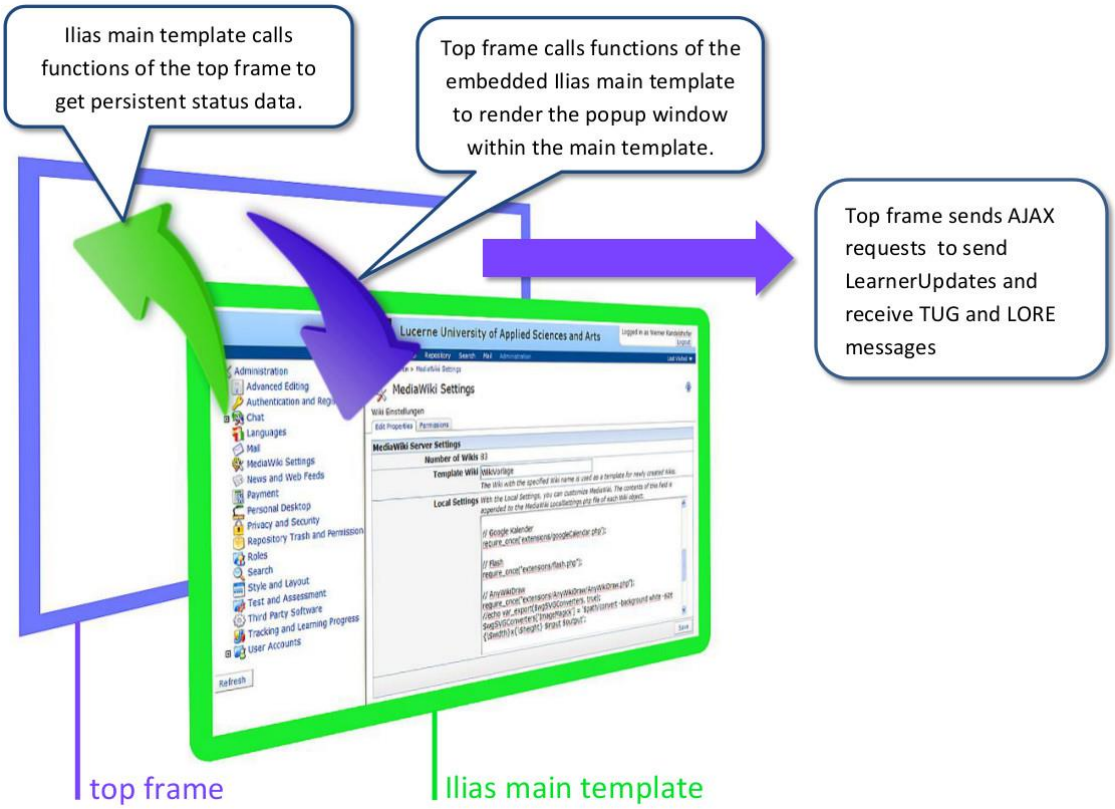

Figure 7: Communication between top frame and main template

This JavaScript code is included in the main template of the ILIAS skin as well as in the top frame. Figure 7 illustrates how these JavaScript functions work together. The top frame provides all the JavaScript functionality that refers to the data exchange between the browser and the LMS. More precise, functions contained by the top frame are those functions that are needed to establish the connection between the browser and the REST interface of the INTUITEL integration component. In Figure 1, this refers to the messages 1,2,3,8 and 9.

On the contrary, the main template of the ILIAS skin contains all the functionality that is needed for presenting the according INTUITEL messages to the user. As shown in Figure 1 and Figure 7, the top frame triggers an AJAX request to the Communicator as soon as the user enters a new Learning Object (message paths 1,2 and 3 in Figure 1).

The top frame holds a JavaScript object containing the id of the Learning Object the user has visited before. With every reload of the ILIAS page the id of the currently visited object is retrieved from the URL parameters. If the current id and the previously visited id are not the same, a new LearnerUpdate message is triggered from within the top frame.

As soon as this request is satisfied by a response from the Communicator (messages 8 and 9 in Figure 1), the INTUITEL popup window is presented to the user by calling a JavaScript function in the main template from within the top frame.

Vice versa, the JavaScript code within the main template also calls functions that are located in the top frame in order to get information about the following states:

- Title of the last message
- Content of the last message
- Display status saying if the user has put the window aside

Indeed, all the JavaScript code could also be located only in the top frame. But separating communication functions from display functions, locating the first ones in the top frame and locating the second ones in the main template of the ILIAS skin allows the user to deactivate the presentation of INTUITEL messages without deactivating the entire INTUITEL functionality. When the user deactivates the INTUITEL skin, all the JavaScript functions in the main template are removed whereas the JavaScript functions in the top frame are still present and keep producing and sending data for the INTUITEL reasoning process – no matter if the INTUITEL skin is activated or not.

AJAX and "Same Origin Policy"

AJAX implements a security mechanism that forbids cross domain requests. Practically, this means that the above described communication between the browser and the Communicator of the integration component does not work if the Communicator is located in a domain being different from the domain where the ILIAS system is located.

For this reason, it is assumed that both the integration component and the ILIAS system are located in the same domain. In principle, it is quite possible to locate the ILIAS system and the integration component on two different servers as the integration component is coupled to the ILIAS system only by the ILIAS database which can be easily accessed remotely.

Anyway, if one decides to run the ILIAS system and the integration component on two different servers, other communication methods than AJAX have to be considered. For example, using JSONP instead of AJAX might provide an appropriate solution. But, since JSONP does not support anything else then HTTP-GET requests, this conflicts with the intention to provide a RESTful API using also POST requests.

3.1.9.3 Modifications of the ILIAS source code

The following lists the modifications that are made to the ILIAS source code.

- In the directory <*ILIAS web root*> an "index.html" file is added containing the frameset for the top frame.
- The original file "index.php" is renamed to "index_embed.php" and referred by the top frame in "index.html"
- In the directory *<ILIAS web root>/intuitel* a JavaScript file "intuitel.topframe.basic.js" is added. This file contains all functions that are in charge of connecting to the Communicator, sending and receiving INTUITEL messages and holding persistent data that are needed by the

functions of the ILIAS template to render the INTUITEL message popup. The "intuitel.topframe.basic.js" is included by the "index.html" page.

- In the directory *<ILIAS web root>/Services/JavaScript/js* the "Basic.js" file is extended by a small function "enforceTopFrame" which enforces loading the top frame if it is not present. This becomes important when – for example – the user navigates from the login page to the ILIAS page or if he follows a direct link to an ILIAS page that does not refer to the index.html into which all ILIAS pages should be embedded. In the normal case, this would load the according ILIAS page into the browser without being embedded into the topframe page "index.html". As a result, the JavaScript code that is needed for sending and receiving messages would not be accessible from within the ILIAS pages. Therefore, the user must be redirected to the "index.html" page
- The directory *<ILIAS web root>/Customizing/global/skin/intuitel* contains the template files for the INTUITEL skin. It is mainly a copy of the default ILIAS skin with the following modifications:
    - A JavaScript file "intuitel.message.popup.js" is added, containing all the functions that are needed to render the INTUITEL message popup
    - The "tpl.main.html" file containing the ILIAS main template is extended with a one-line inclusion of "intuitel.message.popup.js"
    - A modified CSS file is included with style sheet information for the INTUITEL message popup

The above list illustrates how minimal the changes to the original ILIAS source code are. As far as only the ILIAS GUI is concerned, except for the addition of some JavaScript files, modifications of the ILIAS source code are limited to one-line statements including those JavaScripts. All modifications having to do with the presentation of messages in the ILIAS GUI are injected into the ILIAS source code by those JavaScripts on runtime. Actually, not a single line of the ILIAS source code is modified. This makes this solution easily maintainable and extendable. Moreover, checking for transitions between Learning Objects as well as sending and receiving INTUITEL messages is done completely by the browser. This disburdens the integration component and makes it thus more scalable.

However, this does not include the provision of particular information from the database. As there may be some data needed by INTUITEL that are not present in the ILIAS database some modifications of the ILIAS database as well as of some PHP files communicating with the database may be necessary.

3.1.9.4 Parameters and Contents of INTUITEL Messages

As partly mentioned earlier, the XSD definition for LearnerUpdates, TUG and LORE messages are as follows.

```
<INTUITEL>
   <LearnerUpdate mId=" uuid" uId="user ID" loId="LO ID"
    time="access time"/>
</INTUITEL>
```

Codelisting 2: LearnerUpdate

```
<INTUITEL>
   <Tug uId=" user ID" mId=" uuid" [rId="uuid"]>
      <MType>message type</MType>
      <MData>message data</MData>
   </Tug>
   <!-- Repeated if necessary -->
</INTUITEL>
```

Codelisting 3: TUG message from INTUITEL

```
<INTUITEL>
   <Lore uId="user ID" mId="uuid" [rId="uuid"]>
      <LorePrio loId="some LO id" value=" value between 1 and 100"/>
         <LoreLoName>name of the lo</LoreLoName>
         <LoreLoDesc>description of the lo</LoreLoDesc>
      </LorePrio>
      <!-- Repeated if necessary -->
   </Lore>
   <!-- Repeated if necessary -->
</INTUITEL>
```

Codelisting 4: LORE message from INTUITEL

The attributes of the above listed messages are explained in the following.

Message Id (mId)

Every message must be assigned a unique message id. As there is consensus that this id must conform to the uuid standard, the browser-side JavaScript functions can create such message ids on their own, independently from other components.

User Id (uId)

Besides a unique message id, in all cases a unique user id is required. Normally this user id equates to the according id assigned by the ILIAS system. However, the pure JavaScript solution presented above, runs only within the client. Therefore, the ILIAS user id is not available for the creation of a LearnerUpdate message. Instead, we must make use of the session id, which – in the case of ILIAS – is provided by a cookie.

Consequently, the browser sends session ids as uIds to the Communicator of the integration component when emitting LearnerUpdates. According to Figure 1, this message is passed to the

Broker which passes it to the Core module. The latter one finally is responsible for the management of learner updates.

The LearnerUpdate module must then map the session id to an ILIAS-internal user id by accessing the DB-connector which resolves the desired user id by the given session id stored in the ILIAS database. So far, this describes messages from the browser to the integration component. Conversely, according responses from the integration component to the browser need to be supplied with a session id again. Hence, in this case, the respective Core modules must invert the previously described id mapping.

Mapping a session id to a user id is always unique whereas the opposite way may lead to ambiguous results since a user may be associated with more than one session id. To solve this problem, the Core modules must store pairwise assignments of message ids and session ids. When a response appears, this assignment is used to resolve the session id belonging to the corresponding message id. This way, the resolution of session ids from user ids becomes unique too.

Learning Object Id (loId)

The Learning Object id used in INTUITEL messages is equivalent to the ILIAS-internal object id that identifies courses, tests, wikis, multimedia objects and so on. In the case of ILIAS, this id can be directly gathered from the URL parameters of the currently loaded page which is implemented by the JavaScript functions included in the top frame as illustrated by Figure 7.

Access Time (time)

The time when a learner accesses a Learning Object is determined by a JavaScript function locally in the browser.

Name and Description

The name and description field are necessary for the presentation of learning recommendations to the user. The information for this is kept by the ILIAS database from which the LORE core module must retrieve it using the DbConnection module.

3.1.9.5 Packages and Interfaces

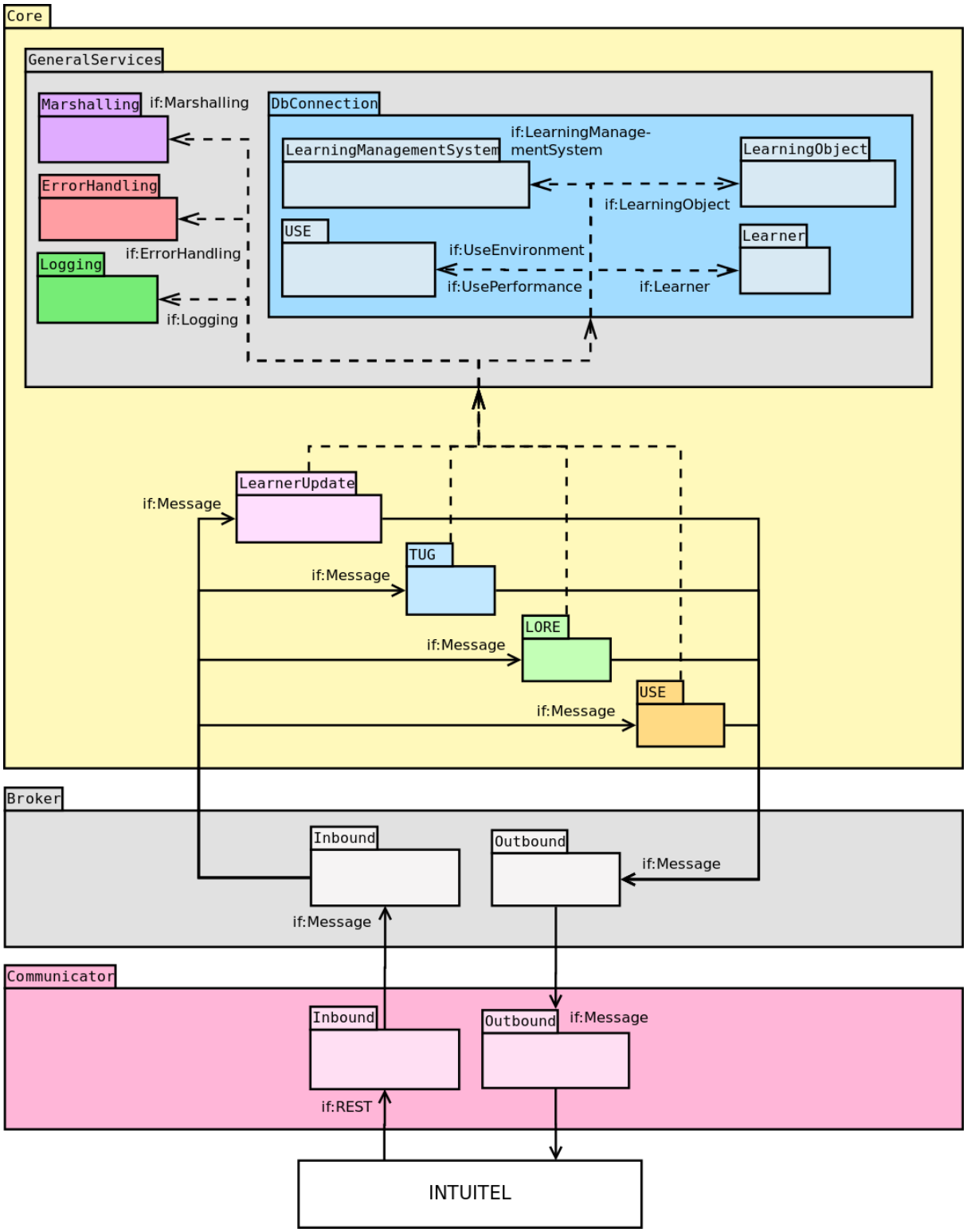

Figure 8: Packages, interfaces and dependencies

This section refines the component model described in the previous section by defining packages and the interfaces they implement. As mentioned at the beginning, the Broker as well as the Communicator is split into two parts: an inbound module and an outbound module. This satisfies the condition that there must not be any double coupling of modules. Figure 8 shows the concrete definition of packages and the dependencies between them. Note that due to the avoidance of double coupling Figure 8 shows an acyclic graph.

In favour of clarity, the methods of the listed interfaces are not documented in detail here. Anyway, the notation of the methods are self-explanatory to the greatest possible extend.

The DbConnection package is contained by its parent package GeneralServices and is subdivided into the packages LearningMangementSystem, LearningObject, Learner and USE. The interfaces of these sub-packages are used from within the LearnerUpdate, TUG, LORE and USE sub-packages of the Core package to access the Ilias database.

The Broker package encloses two sub-packages to separate the outbound communication from the inbound communication. It has thus a specific inbound and an outbound package, each implementing the Message interface. The outbound package's task is only the delegation of messages it receives at its Message interface to the according Message interface of the Communicator.

However, the implementation of the Broker's inbound package is more complex. It is in charge of connecting the Broker to the following Core packages: LearnerUpdate, TUG, LORE and USE. As shown in Figure 8, the Broker's inbound package depends on only one interface – the Messageinterface that is implemented by each of these Core packages.

Just like the Broker, the Communicator package includes an inbound and an outbound package. The inbound package provides a REST-API accessed by the INTUITEL system and the user's browser. On the other side, the inbound package uses the Message interface of the Broker to delegate those requests to it. The outbound package implements the Message interface that is used to delegate messages from the Broker to the Communicator.

The particular interfaces and their signatures are defined in the following.

If:LearningManagementSystem

Package: eu.intuitel.hska.core.generalservices.dbconnection.lms

The LearningManagementSystem interface provides general information referring to the Learning Management System as a whole. This comprises the quantity of learners and session data.

Provided methods:

- int selectNumAllLearners()
- List<String> selectLearnersByStatus(Boolean online)
- int selectNumLearnersByStatus(Boolean online)
- List<String> selectSessionsByLearner(String uId)
- String selectLearnerBySession(String sessionId)

If:LearningObject

Package: eu.intuitel.hska.core.generalservices.dbconnection.learningobject

This interface yields all information that is associated with a specific learning object. This for example includes technical aspects like size and colour mode, the media type or language settings. Furthermore, relationships to other objects like parents and siblings are considered.

Provided methods:

- String selectName(String loId)
- String selectParent(String loId)
- String selectPrecedingSibling(String loId)
- <String> selectChild(String loId)
- String selectFollowingSibling(String loId);
- String selectLanguage(String loId);
- String selectType(String loId);
- XMLGregorianCalendar selectLearningTime(String loId)
- int selectTypicalAgeLowerBoundary(String loId)
- int selectTypicalAgeUpperBoundary(String loId)
- String selectMedia(String loId)
- int selectSize(String loId)
- boolean selectFullCourse(String loId)

If:UseEnvironment

Package: eu.intuitel.hska.core.generalservices.dbconnection.use.environment

The UseEnvironment interface provides data about the technical parameters of the Learning Management System.

Provided methods:

- String selectDisplayType(String uId)
- String selectConnectionType(String uId)
- int selectConnectionStability(String uId)
- String selectResolution(String uId)
- int selectBattery(String uId)
- int selectNoiseLevel(String uId)
- XMLGregorianCalendar selectLocalTime(String uId)

If:UsePerformance

Package: eu.intuitel.hska.core.generalservices.dbconnection.use.performance

This interface provides functionality that refers to performance data of a single user. This refers to the states of learning objects a user is associated with.

Provided methods:

- int selectGrade(String uId, String loId)
- int selectCompletion(String uId, String loId)
- int selectSeenPercentage(String uId, String loId)
- boolean selectAccessed(String uId, String loId)

If:Learner

Package: eu.intuitel.hska.core.generalservices.dbconnection.learner

The Learner interface comes with methods that yield personal data about a user like name, age and so on. Besides this, data about the user's learning objects as well as about the user's online status is provided.

Provided methods:

- ILearner selectLearner(String uId)
- String selectGender(String uId)

- String selectName(String uId)
- int selectAge(String uId)
- String selectCulture(String uId)
- List<String> selectVisitedLearningObjects(String uId)
- List<Map<String, Object>> selectVisitedLearningObjectsWithTime(String uId)
- boolean isLearnerOnline(String uId)
- boolean isLearnerRegistered(String uId)

If:Marshalling

Package: eu.intuitel.hska.core.generalservices.marshalling

The Marshalling interface is used by the Core modules (LearnerUpdate, TUG, LORE, USE etc.) to transform the XML messages they get from the Broker into Java objects and vice versa.

Provided methods:

- String marshallIntuitelXML(INTUITEL intuitel)
- PageObject unMarshalPageObject(String xmlString)
- INTUITEL unmarshalIntuitelXML(String xmlString)
- XMLStreamReader createXMLStreamReaderByString(String xmlInput)
- setNextElementReader(XMLStreamReader xmlreader, String elementname)

If:ErrorHandling

Package: eu.intuitel.hska.core.generalservices.errorhandling

Just like the Marshalling interface, the ErrorHandling interface acts as a service that is consumed by all Core modules. It's task is the provision of a uniform interface for error handling and error report. All exceptions occurring in the context of a Core module are delegated to the ErrorHandling interface. The ErrorHandling module then classifies the Errors and decides how to handle them.

Provided methods:

- boolean setOutputStream(OutputStream oStream)
- boolean setOutputFile(String filePath)
- boolean putError(int errNo, String errMessage)
- boolean setCallbackObject(ErrorCallback callback)

If:Logging

Package: eu.intuitel.hska.core.generalservices.logging

The Logging interface is in charge of handling all non-error events and log them either to a database or text files. Similar to the ErrorHandling interface, all log events occurring within the Core modules are delegated to this interface. The log messages coming from the core modules are classified by the Logging module which decides how to handle those log messages.

Provided methods:

- boolean setOutputStream(OutputStream oStream)
- boolean setOutputFile(String filePath)
- boolean setLogLevel(int)
- boolean putLogMessage(int level, String logMessage)
- boolean setCallbackObject(LoggingCallback callback)

If:Message

Package: eu.intuitel.hska.message

The Message interface includes only one generic method for the delivery of LearnerUpdate, TUG, LORE, USE messages etc. It is implemented by the LearnerUpdate,TUG, LORE and USE package of the Core package. Furthermore, it is implemented by the Broker's inbound and outbound package as well as by the Communicator's inbound package:

Provided methods:

- deliverMessage(String xmlMessage)

The reason why this method is generic on the one hand is that both the Communicator and the Broker are supposed to be totally unaware of the message contents as they are only responsible for delivering messages to the next layer. However, the Broker at least has to know about the type of the message (LearnerUpdate, TUG, LORE, USE, etc.) in order to delegate them to the according Core module. Therefore the Broker looks at the headers of the messages but it does not consider the body of the message. The second reason for providing a generic interface is that it makes it easier to add, modify or remove message types if needed.

The last level of message delivery is represented by the Core modules LearnerUpdate, TUG, LORE, USE etc. These modules also receive messages over the Message interface. Finally these Core modules are the ones that marshal the XML-based messages to Java objects in order to process them. For this purpose, they use the Marshalling interface as described above.

If:REST-API

Package: eu.intuitel.hska.communicator.inbound.rest

The REST-API populates the functionality of the entire integration component to the INTUITEL system and to the user's browser. It therefore belongs to the inbound package of the Communicator and provides the following services in the form of URLs.

Provided methods:

- /intuitel/lmsprofile
- /intuitel/learners
- /intuitel/mapping
- /intuitel/login
- /intuitel/tug
- /intuitel/lore
- /intuitel/use/performance
- /intuitel/use/environment

The REST-API can be understood as the counterpart of the Message interface as it receives HTTP requests that are directly mapped to the correspondent methods of the Broker's Message interface. Hence, the Communicator's inbound package serves as a translator from HTTP requests to the Message interface.

3.1.10 Verification

The USE/TUG/LORE integration in ILIAS has been externally verified. This means that the development and the verification tasks were executed by different teams, working at different institutions. This approach ensures an independent testing, and also checks operability in cross-domain situations.

The external verification has followed a functionality oriented approach. That is, the expected functionality of the developed integration has been stated as a set of test cases, developed in coordination between the developers and the verifiers. Each of those test cases describes the input received by the system, and the expected system response. The verification process consists in comparing the expected response with the actual one.

According to the developed functionality, the verification methodology is divided in two main parts: firstly, the REST interfaces that connect ILIAS with the INTUITEL Engine; secondly, the Graphical User Interface offered to the end-user (the student).

3.1.10.1 REST interface

The developed tests cases are focused on the validation and completeness of the exchanged XML messages. The verification approach is based on the testing methodology of the Communication Layer. This section summarizes the procedure of the testing plan, and further details can be consulted at Deliverable 3.3.

The test cases are programmatically achieved. That is, a piece of software automatically compounds and sends the REST messages, receives the response and checks its correctness. PHP is the programming language used for the implementation of testing suite. In particular, the PHPUnit[5] libraries have been used. According to the provided license terms, the redistribution and use of PHP unit are permitted under some conditions that are satisfied by the INTUITEL project[6].

Developed tests

The external nature of the performed verification implies that internal procedures and communication among subsystems have not been tested. In practice, the test cases cover the communication between the LMS and the INTUITEL Engine, but do not test the communication between the browser and the LMS. The REST interfaces have been verified for those messages in which ILIAS behaves as a server. That is, the testing suite verifies the ILIAS response to the following message types:

- USE Request
- LORE Message
- TUG Message

Table 1 presents the developed tests for each of the above listed message types:

Test name	Description	Expected result
messagetype_001.1	Send valid-simple request and check the HTTP response code	HTTP Status: 200
messagetype _001.2	Send valid-simple request and check the well formation of the response	Well-formed XML received in the payload
messagetype _001.3	Send valid-simple request and check the validity of the response	Valid XML received in the payload
messagetype _004	Send valid request, with different content-type in POST header	HTTP Status: 4XX (bad request)
messagetype _005	Send request with wrong root	HTTP Status: 4XX (bad request)

[5] http://phpunit.de
[6] https://github.com/sebastianbergmann/phpunit/blob/master/LICENSE. See also Deliverable D11.5

	element name (INTUITEL)	
messagetype _006	Send request with wrong main element name (Use / Tug / Lore)	HTTP Status: 4XX (bad request)
messagetype _007	Send request with wrong attribute names	HTTP Status: 4XX (bad request)
messagetype _008	Send request with empty attribute values	HTTP Status: 4XX (bad request)
messagetype _009	Send request with empty payload (no XML sent)	HTTP Status: 4XX (bad request)

Table 1: Developed test cases for each message type

Result reports

A result report will be automatically generated, stored and available at http://tel.unir.net/intuitel-tests/

For each of the test cases, the report contains the information presented in Table 2.

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 2: Template for test case reports

The initial iterations of the tests cases revealed a number of issues in the REST interfaces. For example, case sensitivity for the URL and HTTP headers support has been modified in the server due to failures detected with the test suite.

The test suite is periodically executed, and the result reports are accordingly updated. The error report is therefore a live document that is frequently changed. The reports, and the test suite documentation is available at http://tel.unir.net/intuitel-tests/

3.1.10.2 Graphical User Interface

In the case of the Graphical User Interface, the verification is human oriented. That is, the test cases consist of a sequence of actions to be performed in the browser by the end user. The goal is to check whether or not the result presented in the screen is the expected one.

The verification of the Graphical User Interface is therefore manually performed. A more sophisticated approach, including automated GUI testing will be performed in the scope of WP12, within the artificial test learners.

In order to ensure browser interoperability, the verification of the ILIAS GUI needs to be executed in the main browsers existing in the current market (Internet Explorer, Mozilla Firefox and Google Chrome). The GUI testing depends on the availability of a demo ILIAS server and it is therefore a work in progress. It is planned to be executed once the demo server is available, and the result report will be accordingly generated.

Use cases

The test cases will focus on the following aspects:

Appearance:

- Does the popup window look the same in different browsers?
- Does the movement of the popup behave correctly in different browsers?
    - Slide in / slide out
    - open / close window

LearnerUpdate/TUG/LORE Messages

- Does the popup react correctly due to message sequence LearnerUpdate → TUG → LORE?
    - Browser omits LearnerUpdate, TUG and LORE requests when the user opens a new learning object.
    - Messages are only shown, when asynchronous TUG and LORE responses are both received by the browser. In the meantime, the browser keeps on waiting, even if only one of the two messages is received.
- Does the popup behave the same according to different chronological orderings of response messages? (ordering must be arbitrary)

Navigation Within and Between LOs

Scenario 1:

- User opens LO (LearnerUpdate, TUG, LORE requests are sent by the browser)

- User navigates to another page within the LO.
- TUG/LORE responses arrive
- Does the popup appear correctly although user has moved to another LO subpage?

Scenario 2

- User opens LO (LearnerUpdate, TUG, LORE requests are sent by the browser)
- User navigates to another LO (new LearnerUpdate, TUG, LORE requests are sent by the browser)
- new TUG/LORE responses are received
- Does the popup show correctly the new TUG/LORE contents according to the new LO?
- Are the old messages discarded?

Scenario 3

- User opens LO (LearnerUpdate, TUG, LORE requests are sent by the browser)
- User navigates to another LMS page that is not an LO
- TUG/LORE responses are received
- Are theses messages discarded or shown to the user either?

3.1.11 Conclusions

The approach described here emphasizes a lightweight integration of INTUITEL functionalities into the LMS. The presentation of TUG/LORE/USE messages is entirely decoupled from the LMS since only JavaScript is used to inject message presentations into the ILIAS web interface. This technique could hence be used for other LMS.

The communication between the LMS and INTUITEL as well as the communication between the LMS and the user's browser is entirely decoupled from those parts that are responsible for the processing of TUG/LORE/USE etc. messages. This is due to a layer-based architecture that may also be used within other LMS. If used for other LMS, neither the architecture nor the interfaces do need any changes. Only the inner logic of the respective Core modules has to be adapted. The loose coupling and the fact that only few modifications must be made to the LMS sources lowers the barriers for LMS vendors to integrate INTUITEL functionality into their LMS systems.

The message format for the communication between the user's browser and the LMS is chosen equivalently to the data model described in Deliverable 1.1. Note that this part is completely invisible to the INTUITEL system as this concerns only the communication between the browser and the LMS. We could even choose a completely different data model. However, we decide to use the already

existent data model from Deliverable 1.1 since introducing a new data model would make things unnecessarily more complicated and thus more difficult to understand. Moreover, the overall system design becomes more cohesive.

3.2 USE/TUG/LORE in Moodle (UVA)

Moodle is a pure-HTTP platform developed with accessibility and portability in mind. This implies that Javascript is used with the principles of "unobstructive JavaScript" and "progressive enhancement" in mind. Efforts to allow full functionality without JavaScript are very encouraged in the Moodle Development Guidelines[7].

Moreover, Moodle has several extension mechanisms that can be used to add code to different stages of the cycle of life of page generation. The relevant to our goals are:

- **Blocks**: Plugins that render a box with contextual content. They are configurable to be embedded besides the main content of many pages. They allow to enrich the page contents and to include custom code while generating a page at the server. This is the mechanism of choice to implement the communication with the student and to trigger notifications of events to INTUITEL services from the LMS (see Figure 9).
- **Modules**: Plugins intended for pedagogical content and activities. Most of the implementation of the modules is beyond the control of Moodle although some functions are mandatory such as "log_access". The Logging API allows modules to add new entries to the Moodle log and define how they get displayed in reports. Logging is an extremely important aspect of Moodle Plugin development. All important actions such as viewing, deleting, editing etc. should be logged.

Although there is a login/logout mechanism, usually there is no information about whether as user is effectively logged-in or not. With respect to INTUITEL, there is not any communication channel to the student that can be started by the LMS; all information should be pulled by the student's browser. Hence, the Moodle adaptor relies on the persistence of the TUG and LORE messages in the Intuitel server to function. This approach avoids dealing with messages timing and sequencing and takes advantage of the return channel of LearnerUpdate requests for getting TUG and LORE content.

From the external services point of view, there is no problem to implement REST service points because every script in the PHP server acts as a potential REST service. Every access to these services is checked against a white-list of allowed remote clients. The list is maintained by an administrator in the global settings of the Moodle platform. All parameters used in SQL queries are filtered using a Moodle API to avoid SQL injection attacks. This behaviour fulfils the security requirements of INTUITEL system.

[7] http://docs.moodle.org/dev/Coding

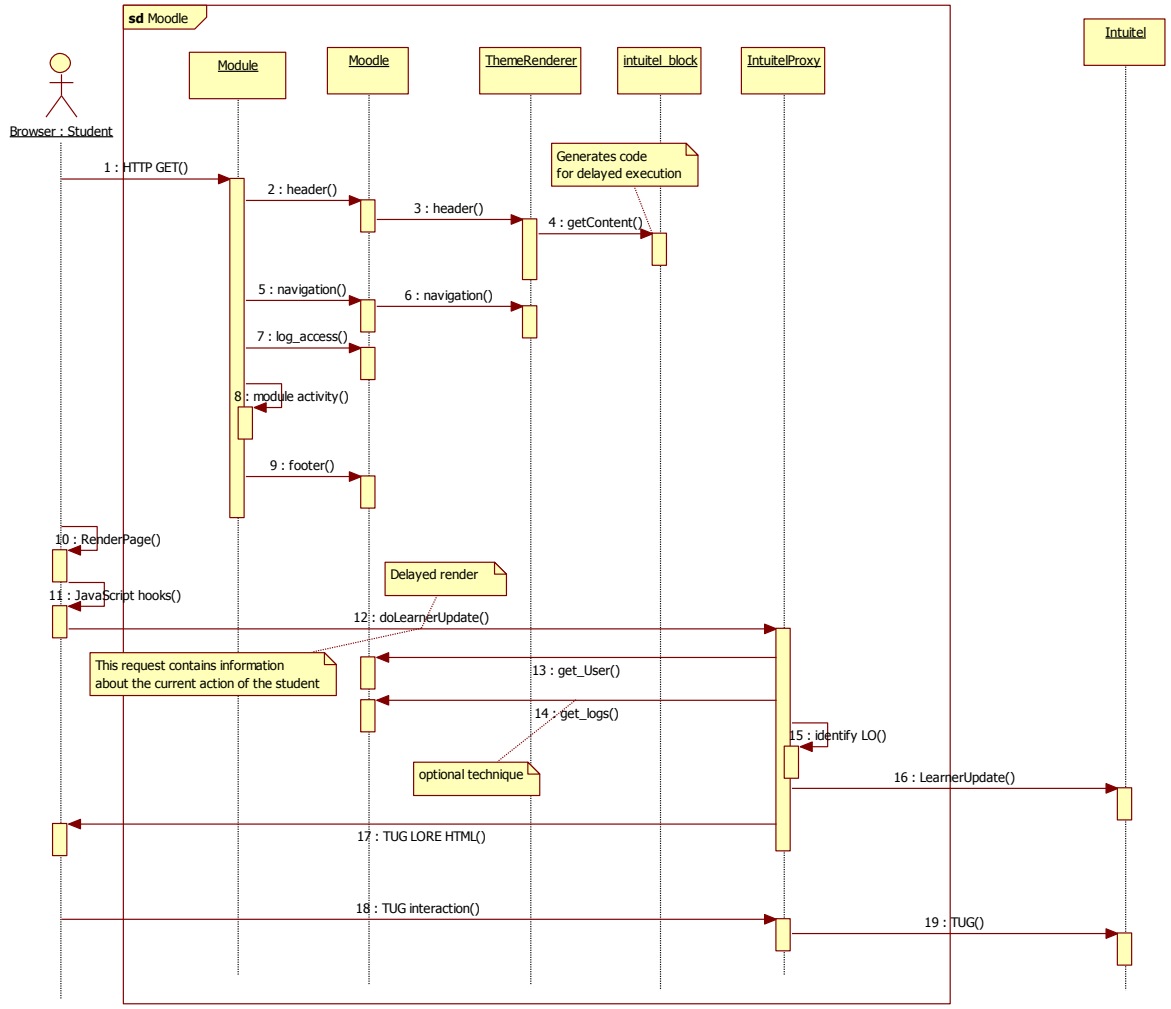

Figure 9: Schematic interaction diagram for the monitoring mechanism based on the "Blocks" extension method of Moodle

3.2.1 Architecture of the implementation for Moodle

Being implemented as a block of Moodle, the source code of Intuitel interfaces must be placed in the folder "blocks" of Moodle within a folder called "intuitel": [MoodlePath]/blocks/intuitel.

The source files of the block *intuitel* are organized into different folders:

/intuitel: root directory of the block in which files implementing entry points to the REST web services are available:

- auth.php: Serves delegated authentication requests (see Figure 9).
- learner.php: Serves INTUITEL requests for learners' information.
- lmsprofile.php: Provides a set of metadata about the capabilities of the Moodle Adaptor for INTUITEL.

- lore.php: LORE requests (see Figure 9).
- mapping.php: Describes the structure of courses and learning objects (see Figure 9)
- tug.php: TUG requests (see Figure 9).
- use_env.php: Provides available environmental information about the user (see Figure 9)
- use_perf.php: Provides information about users' scores in the different learning objects (see Figure 9).
- rest.php: This script redirects requests to the proper script depending on the URL format to adhere to the communication layer REST specification of INTUITEL. Hence base URL for Moodle (<INTUITELEndPoint> in D1.1, section 3.1) has the form: http://hostname/moodledir/blocks/intuitel/rest.php

Besides, in this folder there are files required for any block of Moodle :

- version.php: contains definitions needed by Moodle in the install/upgrade process such as the version of the block and the lower version of Moodle required to use it.
- block_intuitel.php: this holds the class definition for the block, which is an extension of Moodle class *block_base*, including attributes and methods for managing the block and rendering it on the screen. For browsers supporting *JavaScript* an asynchronous *AJAX* request is used, while for the others either *iframe* is used or the content is inserted directly in the page. The later implies an increased delay while loading the page. By default, *iframe* is used as set in the configuration option *'block_intuitel_no_javascript_strategy'* defined in the file setting.php .
- settings.php: to enable global configuration of the block, that is, configurations done by the administrator that apply to every instance of the block.

And some files specific for the Intuitel interfaces:

- module.js: Holds the JavaScript (following YUI arquitecture) for the interaction with the page.
- IntuitelProxy.php: takes the requests from the block and forwards them to the Intuitel service. At this point the user is ensured to be present in the platform and the current interaction is guaranteed to be on going.

/intuitel/model: contains the files defining the different classes of the general model, which is applicable for any LMS.

/intuitel/impl/moodle: contains the files with the specific implementation of the interfaces for the LMS Moodle. Therefore, classes defined in these files are specializations of the general ones in the folder "model".

/intuitel/db: contains the file install.xml with the definition of the database tables needed by the block, which are created when the block is installed.

/intuitel/lang: holds internationalization files.

/intuitel/pix: contains the icons and images used by the block.

/intuitel/scripts: files needed by the used *JavaScript* libraries.

/intuitel/tests: for testing purposes, contains the test cases used to test and validate the code.

Figure 10 and Figure 11 show the main components of the architecture of the implementation of the interfaces for Moodle. In order to allow the reusability of the model in future implementations or for other LMSs, the classes in intuitelLO.php in Figure 11 and the ones shown in the frame "Intuitel Model" in Figure 10, define the needed data structure, and define, and/or implement, all the operations that allow any LMS to operate with INTUITEL. The classes included in the "Moodle implementation" frame are specific to Moodle, as well as all descendant classes of intuitelLOFactory, specializing the methods to adapt the data structure and operations to Moodle.

Below, the classes of the INTUITEL model are briefly described:

- IntuitelController: Holds the logic for the INTUITEL protocols when processing the xml messages received through each REST interface.
- IDFactory: an instance descendant of this class is in charge of generating and converting unique identifiers for Learning Objects, Users and Messages. IdFactory defines the methods for mapping the learning object ids used by Intuitel (and provided by the LMS), with those native ids used by the LMS. The descendant class MoodleIDFactory implements those methods for the specific Moodle implementation. Instances are built by means of a registered Factory object.
- IntuitelLO: parent class for all INTUITEL Learning Objects, having as parameters all those included in table 3 of Deliverable 1.1 (eg. loName, loId, etc.). This class is basically a data container and its methods are those to get and set the different parameter values, as well as a utility method called *match* used when searching learning objects by attribute-value pairs.
- LOFactory: this abstract class defines the way to support different content types in any platform. This allows hiding the platform-specific details when constructing a new instance of the class intuitelLO. Currently 25 LOFactories are implemented for Moodle plus one IDFactory.
- IntuitelAdaptor: API for accessing information about learners and courses in the current platform. It is used following the "factory" pattern to allow a potential reutilization of the implementation in other platforms. For Moodle, the implementation impl/moodle/MoodleAdaptor.php is provided. All factories are registered at runtime when including the script impl/moodle/moodleAdaptor.php in the code of the plugin installed at blocks/intuitel.
- IntuitelAdaptorFactory: to create a suitable instance of IntuitelAdaptor to undertake LMS-specific operations (see 3.2.1 Architecture of the implementation for Moodle).

- Intuitel: defines static variables and operations to provide a way to get the proper LOFactory for a native representation of content in the platform, as well as the proper IntuitelAdaptor object for LMS-wide or course operations.

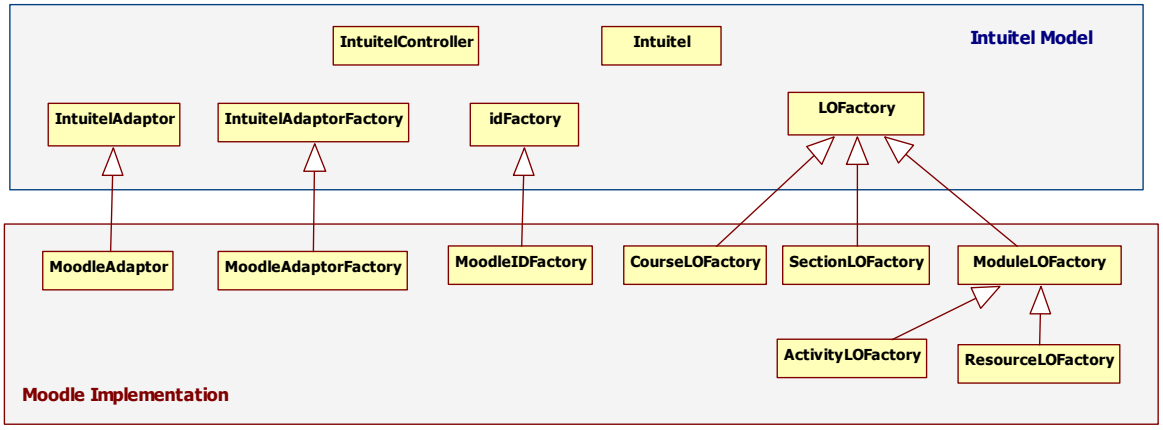

Figure 10: Main components of the "factory" pattern in the implementation for Moodle

As shown in Figure 10, MoodleAdaptor, MoodleAdaptorFactory and MoodleIDFactory are specializations of their respective parent classes needed to undertake the defined operations in Moodle. The different descendants of LOFactory are also needed to create the specific LOs of Moodle.

Moodle courses are typically organized into sections that are considered containers of two types of learning objects (called modules): resources and activities. Resources involve receptive knowledge while activities require actions by the student, and usually have an associated grade. Therefore, the logical data model, shown in Figure 11, has been organized following the structure of a Moodle course. Then, the different specialized descendant classes of IntuitelLO correspond to each particular type of LO in Moodle.

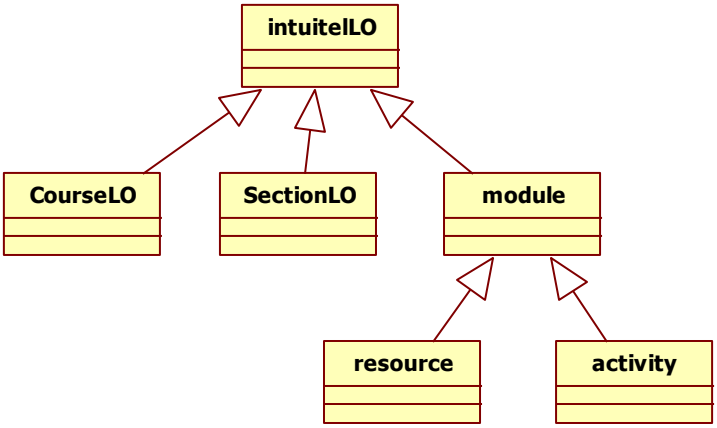

Figure 11: Data Model Diagram

3.2.2 Lmsprofile

Specific capabilities of Moodle instance are specified in the response to an LMSprofile request. The method to obtain the capabilities is implemented in any descendant of IntuitelAdaptor (i.e. class MoodleAdaptor). It simply returns an array of properties used to compose the response, which are listed below:

- `lmsName`: obtained using a core function of Moodle.
- `lNameFormality`: set to value 3, that is, a forename and surname is used to address the user. This is the formality level used by Moodle.
- `lmsType`: statically set to 'Moodle'.
- `lmsId`: retrieved from the global variable *CFG* of Moodle. When installing the Moodle block, the administrator must configure different parameters such as the LMSid which is stored as a global configuration attribute of name 'block_intuitel_LMS_Id'.
- `lmsMediaLevel`: set to 'va' (v- video, a-audio)
- `loreLevel`: set to 1, that is, LORE is not emulated via TUG.
- `useLevel`: set to 1, that is, USE is not emulated via TUG.
- `comStyle` : Not having a mechanism to deliver PUSH messages to the user, comStyle is set to 0 (pull communication).

3.2.3 Learner update

In order to notify INTUITEL about LO transitions, each interaction of the student triggers an event notified to INTUITEL by means of a LearnerUpdate request that expects in return a TUG and a LORE message piggy-backed in the response (see Figure 9). By default, only one event is reported although the implementation allows also reporting a list of events from the event log (by changing the 'block_intuitel_report_from_logevent' configuration option defined in file setting.php).

In addition, a mechanism for INTUITEL to be able to poll for learning transitions has been implemented. As response to the “Learners” request, Moodle provides an XML with a list of learning objects visited since the last poll for each active leaner. For every user, the system registers the last time when the visited LOs have been informed to INTUITEL. As the LMS only responds with the data of active learners, the global configuration parameter “block_online_users_timetosee” is used to decide if a user is active since the last access.

3.2.4 LO mapping and inventory

In order to describe the operations and interactions among the different components involved in the mapping request, a sequence diagram is shown in Figure 12. This corresponds to the simpler case when a Learning Object Id is specified in the mapping request.

After receiving the mapping request the class “Serializer”, responsible for processing the XML data of the INTUITEL mapping message, obtains the different parameters needed to process the request. These parameters are used by an object “IntuitelAdaptor” that implements the needed logic to process the request and provide the response data used by the Serializer to compose the response with XML format.

As the diagram shows, firstly the IntuitelAdaptor object, responsible for the creation of the LO, obtains the type of object corresponding to that loId to instantiate the specific type of LOFactory (ModuleLOFactory in this example). Besides providing the type of LO, idFactory has the important role of mapping the native ids used by Moodle with the corresponding loIds used by Intuitel (and the opposite operation) in this example, as well as in many use cases. For example, in the message 4, the component ModuleLOFactory calls the method getIdfromLoID of idFactory to obtain the native id of the ModuleLO that is being created. The native id is needed to retrieve the native data used to construct the LO by calling some specific function of Moodle as well as some own methods of the factory.

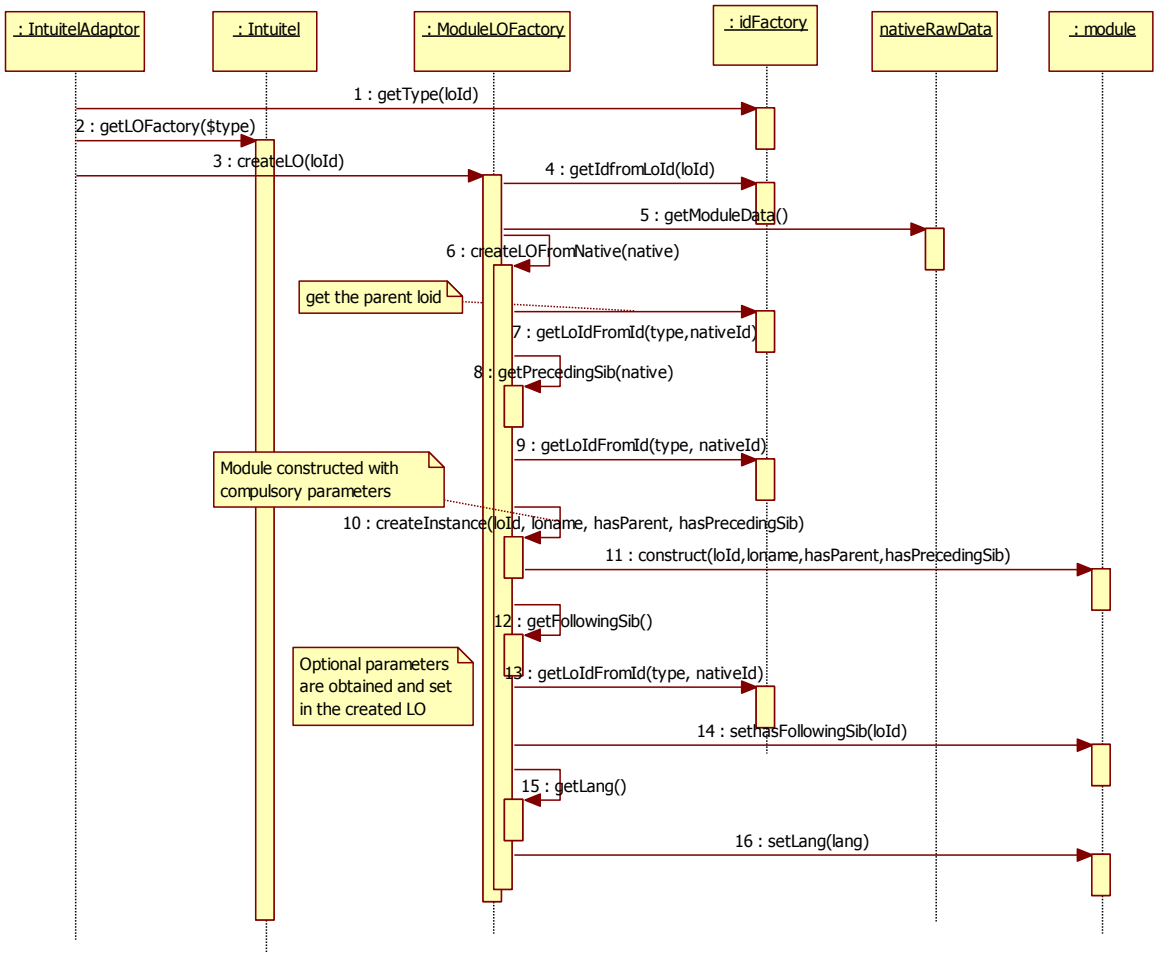

Figure 12: Sequence Diagram to create a learning object

3.2.5 Authentication

The sequence diagram in Figure 13 shows main components and operations involved in the authentication of content creators in Moodle. After obtaining the native user id, the authentication is done by using the authentication methods provided in Moodle libraries. If the user is authenticated, the courses owned by that user are retrieved to provide the data for LoPerm elements of the response. A course is considered to be owned by a user if this user has the capability *block/intuitel:externallyedit,* created to represent the permission of users to edit from external tools. This check is done by the interface IntuitelAdaptor (implemented by the object MoodleAdaptor).

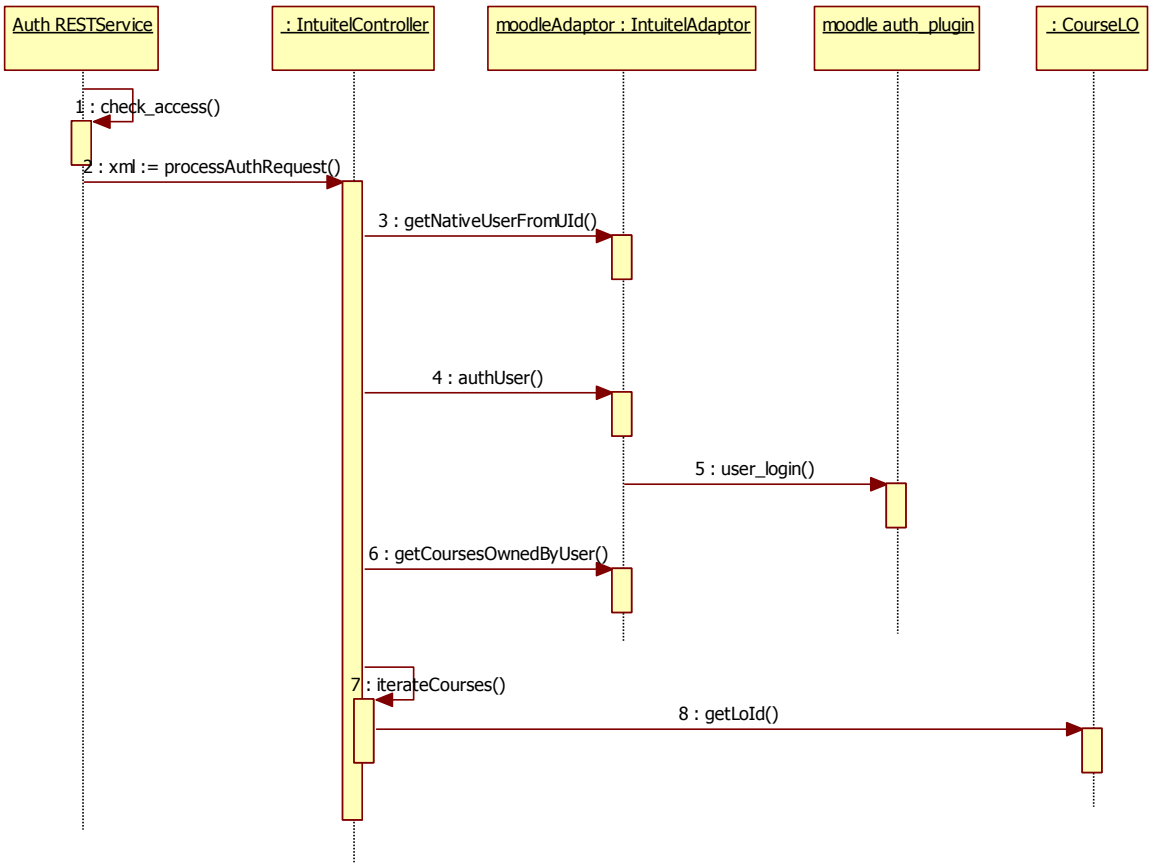

Figure 13: Sequence Diagram with operations for authentication

3.2.6 Tug

As there is no way to deliver any message to the user in response of a TUG request from the INTUITEL server, the only answers are:

- Return an ERROR response message if the user is unknown.
- Return a PAUSE response message to tell INTUITEL that the user is not aware yet of the message but he is known to the LMS and he is expected to visit a page in the future (sooner, later or never).

TUG messages are received as responses from the INTUITEL server to the LearnerUpdate requests sent by the LMS, each time a user enters a new LO (see Figure 9). Therefore, these TUG messages are processed each time the INTUITEL block is loaded in the browser (that is, each time the user accesses a new LO or the main page of the course). The IntuitelProxy class is in charge of forwarding both the LearnerUpdate requests from the block in Moodle to the INTUITEL server, and the TUG requests when a user responses a TUG message displayed in the block.

3.2.7 Lore

As there is no way to deliver any message to the user in response of a LORE request from the INTUITEL server, the only answers when receiving a LORE request from INTUITEL are:

- Return an ERROR response message if the user is unknown.
- Return a PAUSE response message to tell INTUITEL that the user is not aware yet of the message but he is known to the LMS and he is expected to visit a page in the future (sooner, later or never).

LORE recommendations are received as response from the INTUITEL server to the LearnerUpdate request sent by the LMS, each time a user enters a new LO (see Figure 9). Therefore, LORE messages are processed each time the block is loaded in the page (that is, each time the user accesses a new LO or the main page of the course). The implementation displays LORE recommendations within the block content. Besides, by using *JavaScript*, it modifies the DOM model of the page adding stars to the LO addressed (see Figure 14).

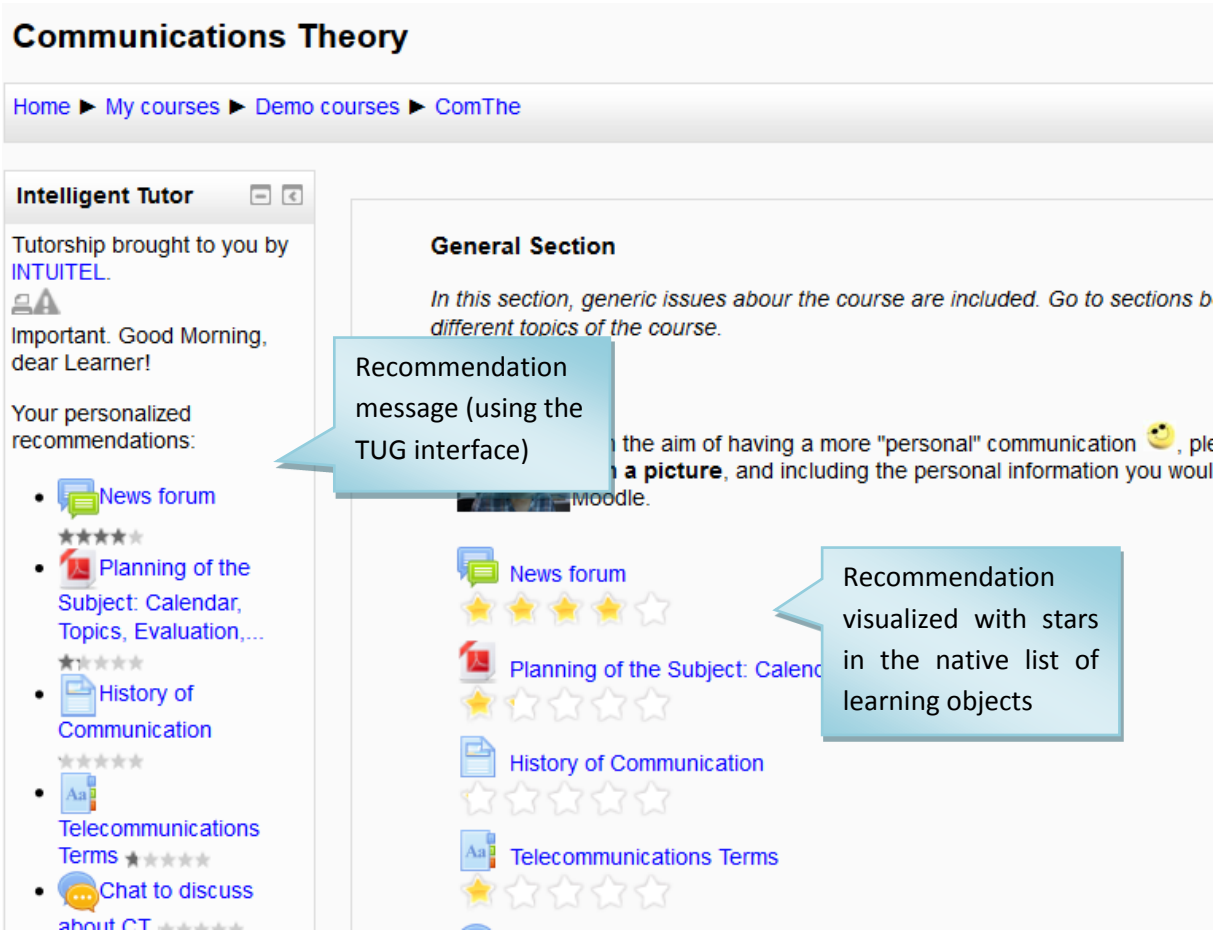

Figure 14: Integration of LORE recommendation into the Moodle course format

3.2.8 UsePerf

The Use Performance request has as response, for each user, a list of scores in each LO contained in every INTUITEL enabled course the user is enrolled in. Figure 15 shows an example of how this operation is undertaken by the block INTUITEL. The example corresponds to the case when the loid attribute is left blank in the request, thus all available learner scores should be informed in the response.

After receiving the UsePerf request, the IntuitelController object will call the method getAdaptorInstance() of Intuitel to create the proper instance of IntuitelAdaptor, that is, a MoodleAdaptor object to undertake Moodle LMS-wide operations. After checking if the user is known (to provide the proper response in another case) and taking the native user identificator, a list of CourseLO objects are provided corresponding to those Intuitel-enabled courses in which the user is enrolled. From this point, the proper MoodleAdaptor to operate within each course is created iteratively. It is in charge of retrieving all LOs of the course and the USE data corresponding to each LO for the corresponding user.

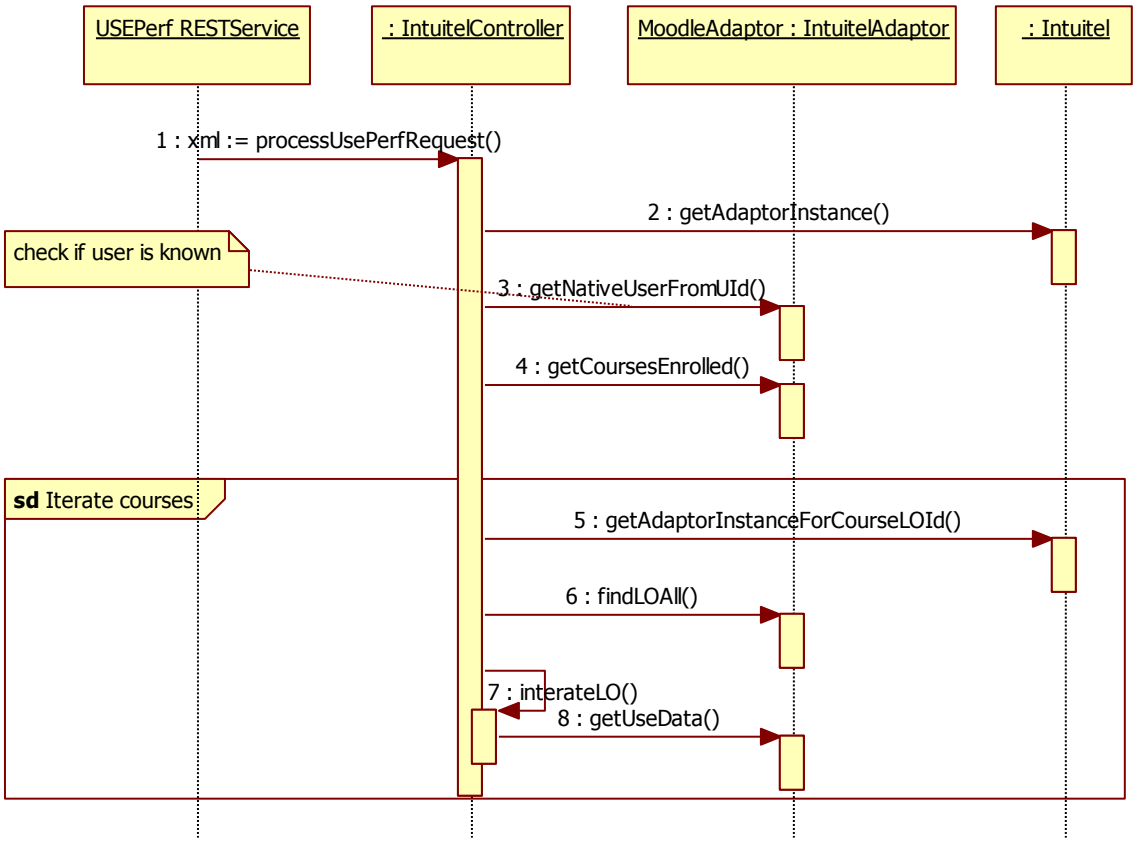

Figure 15: Sequence Diagram: USE Performance Data Request

The operations shown in Figure 15 are frequently used in the different requests. The specific operation for UsePerf request is the method getUseData() implemented by MoodleAdaptor. There are five LO score types, as defined in Deliverable 1.1, and the existence and value of a score type for a learning object depends both on the own nature of the learning object as well as on the configuration of the learning object done by the teacher as explained below:

- **Score type Grade**

In Moodle, the learning objects added to a course are classified into two groups, activities or resources. The first ones are potentially gradable while the second ones are not. The activities can be configured by the teacher to be gradable or not. Therefore, if the LO is configured as gradable and the student has received a grade, the score item will be included in the response for that LO. By calling some methods of the package *core_grades* of Moodle, a list of the gradable learning objects can be obtained by the MoodleAdaptor, as well as the user grades.

- **Score type completion**

Completion tracking for each learning object can be enabled by the teachers in Moodle and a set of conditions can be configured to consider the activity complete. Then, the score type completion will be provided in the response if the teacher has enabled the completion tracking for the activity. The completion status will be 0% or 100% as completion tracking in Moodle does not implement partial completion tracking. Moodle code includes a package called *core_completion* defining the class completion_info that is used to check if completion is enabled for a given LO and to retrieve the completion state.

- **Score type Accessed**

For all LOs this score type is provided in the response, specifying whether a learner has accessed a LO or not. This is implemented by checking logs of user actions registered by Moodle.

- **Score type SeenPercentage**

For LOs whose media type is video or audio, a 0% is indicated in the response for this score type if the object has not been accessed by the user, and 100% in the opposite case. There is no possibility in Moodle to track a partial percentage. This is implemented using the function to get the accessed status.

3.2.9 UseEnv

The response to a UseEnv request reports environmental data of one or more users, the ones whose uId is indicated in the request, only if they are logged in. As with the "Learner update" case, the global configuration parameter “block_online_users_timetosee” is used to decide if a user is active. For each user, after checking that the uId is known and that the user is active, the LMS will respond with three types of environmental data available in Moodle:

- lName: Name and surname of the user.

- dType: Currently used type of device. In order to know the current type of device of the user, each time the INTUITEL block is loaded, it registers this value by using a function provided by Moodle.
- eTime: Current daytime of the leaner
- dLocation: Current location of the user in EWKT (Extended Well Known Text Representation) format (implementation pending on a formal specification by the Intuitel consortium).

3.2.10 Verification programme

A test course in Moodle has been created for verification purposes. A backup of this test course is provided with the code to facilitate future testing. This course contains different users, sections, resources and activities as shown in Figure 16.

The code used for testing is contained in the different files of the folder /intuitel/tests. There is a file for each type of request (eg. mapping_test.php for testing mapping requests) and within each file there are several functions that correspond to a variety of test cases covering different possible requests and asserting that the output is the expected one.

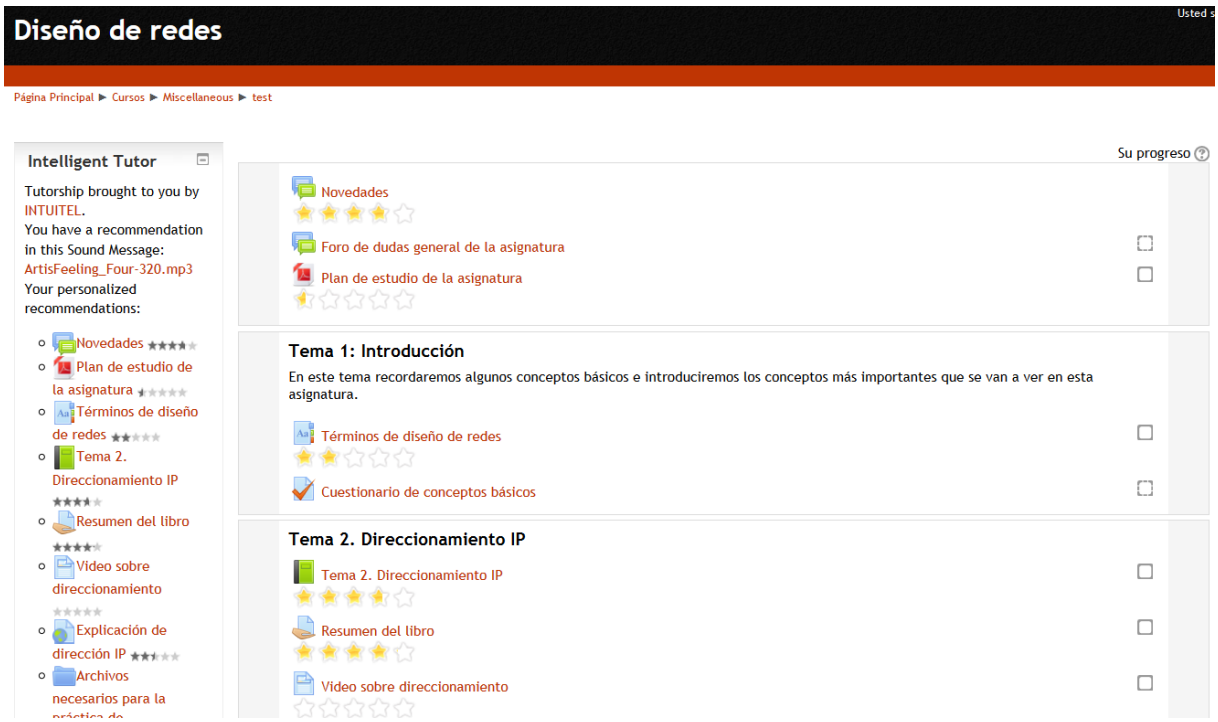

Figure 16:Main page of the course created for testing purposes.

The cases that are tested correspond to these functions:

- Mapping (file /tests/mapping_test.php)

- `test_list_learning_objects_byCourse()`: obtains an array of every Learning Object (LO) in the test course (for testing the special case when a complete course mapping is returned in one response).
- `test_list_learning_objects_byCourseSerializer()`: obtains the XML response to the INTUITEL request specifying the LO ID of the course and the "getFullCourse" parameter.
- `test_list_all_learning_objects()`: obtains an array of every LO in all Intuitel enabled courses, not only in the test course (to test the INTUITEL request that does not include a LO ID nor a list of search criteria).
- `test_find_section_by_LOId()`:obtains the XML response to the INTUITEL request specifying the LO ID corresponding to a section in the test course.
- `test_find_forum_by_LOId()`: obtains an object intuitelLO corresponding to a forum activity in the test course.
- `test_find_by_notexistingLOId()`: to test the case when the LO Id provided in the mapping request does not exist. An exception is launched as result.
- `test_search_course_byname()`: to test the search of a LO corresponding to a course (the search criteria is by 'name' of the LO and with an empty attribute 'hasParent').
- `test_search_lo_byname_and_lang()`: to test the search of a LO by 'name' and 'lang'.
- `test_search_section_byChildren()`: to test the search of a LO by the value of a data item with name 'hasChild'.
- `test_search_lo_by_hasPrecedingSib()`: to test the search of a LO by the value of a data item with name 'hasPrecedingSib'.

- UsePerf (file /tests/use_perf_test.php)
  - `test_push_message_multiple_users()`: this includes two test cases:
    - Case A: the XML request asks for performance data of two specific known LO IDs for two known users.
    - Case B: the XML request asks for performance data of two specific known LO IDs for both a known user and an unknown user.
  - `test_push_message_correct_user()`:the XML request asks for performance data of two specific known LO IDs for a known user.
  - `test_push_message_bad_user()`:the XML request asks for performance data of two specific known LO IDs for an unknown user.
  - `test_push_message_bad_LOId()`:the XML request asks for performance data of a specific known LO ID and an unknown LO Id for a known user.
  - `test_push_message_bad_LOId_User()`: test case in which both the LO Id and the user Id specified in the request are unknown.
  - `test_push_message_allLOIds_User()`: in this case, the loId-attribute is left blank in the request, thus the LMS returns all available LO scores for the learner.

- UseEnv (file /tests/file use_env_test.php)

- `test_push_message_known_user_active()`: the user is known and active.
- `test_push_message_unknown_user()`: the user is unknown.
- `test_push_message_known_user_pasive()`: the user is known but is not active.

- Authentication (file /tests/auth_test.php)
  - `test_push_message_correct_user()`: the Authentication request refers to three users:
    - Known user, teacher of 'test' course, with correct password.
    - Known user with correct password, who is not a teacher in any course (The response has not LoPerm elements).
    - Unknown user.
  - `test_push_message_bad_user()`: the Authentication request refers to an unknown user.
  - `test_push_message_bad_passwd()`: the Authentication request refers to a known user but wrong password.
  - `test_push_message_admin()`: the user Id in the request corresponds to the Admin of the LMS.

- Core (file /tests/intuitel_core_test.php)
  - `test_check_access()`:contains different test cases referred to system authentication. First a not authorized remote Intuitel server is been checked and an exception is launched, later the localhost and an authorized remote server are authenticated successfully.

- Lore (file /tests/lore_test.php)

  LORE recommendations are received as response from the INTUITEL server to the LearnerUpdate request sent by the LMS. In the case a LORE request from the Intuitel server is received, only two types of response are possible which correspond to the following two test cases.

  - `test_push_message_correct_user()`: In this test case the uId of the user is known and the acceptance code (retVal) in the response is equal to PAUSE.
  - `test_push_message_bad_user()`: In this test case the uId of the user is unknown and the acceptance code in the response is equal to ERROR.

- Tug (file /tests/tug_test.php)
  - TUG requests have also only two types of response which correspond to the following two test cases:
  - `test_push_message_correct_user()`: In this test case the user is known and the acceptance code (retVal) in the response is equal to PAUSE.

- `test_push_message_bad_user()`: In this test case the user is unknown and the acceptance code in the response is equal to ERROR.

- Processing of TUG and LORE messages (file /tests/update_pull_response_test).
  - `test_tug_mtype1()`: After receiving the Intuitel response with a MType 1 TUG message, it is tested that the correct message is shown to the user.

    LORE and TUG interfaces have been verified manually. The script /tests/mockrest/intuitel.php implements an emulation of the Intuitel remote service with an example of each type of TUG message and with a random simulation of LORE recommendations.

- Learner Update – Polling (file /tests/learner_test.php)
  - `test_learner_active()`: when the Learners request is received, there are two active users and some LO transitions have not been reported since the last poll.
  - `test_learner_nonactive()`: when the Learners request is received there are not active learners.
  - `Test_learner_active_first()`:when the Learners request is received, there is an active user and LO transitions have never been reported (first poll).

The results of the tests are the expected ones and no remarkable problems have been detected.

3.2.11 Other remarks

3.2.11.1 Pending issues

The format of TUG messages corresponding to mType 4 and 5 are not specified yet. Moodle implementation assumes that W3C form encoding is used.

The geolocation format has not been specified yet. The current implementation uses Extended Well Known Text representation format (EWKT) for geometry.

3.2.11.2 Roles and permissions

In Moodle, a role is a collection of permissions defined for the whole system that can be assigned to specific users in specific contexts (a context is a functional area of Moodle such as a course, a specific activity or a resource...). The combination of roles and context define a specific user's ability to do something on any page.

Although custom roles can be defined, there are standard roles: administrator, manager, teacher, non-editing teacher, student, guest, authenticated users. For these standard roles, specific permissions are established by default although lately these permissions can be modified.

The default capabilities to operate with the INTUITEL block are specified and assigned to the different roles in the file db/access.php and are described below:

- "block/intuitel:myaddinstance" : capability to add an instance of Intuitel block in "My Moodle page". This capability is defined to allow a future implementation.
- "block/intuitel:addinstance" : capability to enable Intuitel in a course (that is, add an instance of the block Intuitel in a course). This capability is enabled by default to the roles teacher, editingteacher and manager.
- "block/intuitel:view": capability to view the Intuitel block in the course, that is, view TUG and LORE messages and interact with them. By default, this capability is set to the roles student, teacher, editingteacher, manager, user, guest.
- "block/intuitel:externallyedit": capability to edit INTUITEL courses from the external editor of INTUITEL. This is checked when processing an Authorization request. By default, this capability is enabled for the roles teacher, editingteacher and manager.

Administrators can change these permissions assigned to each role through the menu "Site administration"->"Users"->"Permissions"->"Define roles".

3.2.11.3 How to configure INTUITEL in a Moodle site

INTUITEL is implemented as a block and therefore it can be installed in a Moodle site as any other plugin of Moodle. Once it is installed, it is important to configure a few different parameters. The administrator of the site can access to the configuration options below described through the menu "Administration->Site administration->Plugins->Blocks->Intelligent Tutor" (see Figure 17):

- Identification prefix for this Moodle instance: This string is used as part of identifiers throughout the INTUITEL system. It is used to ensure that the ids are unique and relevant for this platform. The default value is taken from the unique identification of the Moodle installation, but this can be set to another meaningful value or to retain an old prefix in case the server has been moved and need to reuse existing INTUITEL activities and configurations.
- List of IPs allowed sending INTUITEL events to this LMS: this is a white-list of the hosts allowed to send messages and requests to this instance. Only IPs of trusted INTUITEL servers should be included here.
- Service Point IP for using INTUITEL services: location of the remote INTUITEL server that will produce recommendations for the platform and that is able to receive outgoing messages such as user's interactions and answers.
- The other options of the Developer's section correspond to advanced options and rarely should be changed.

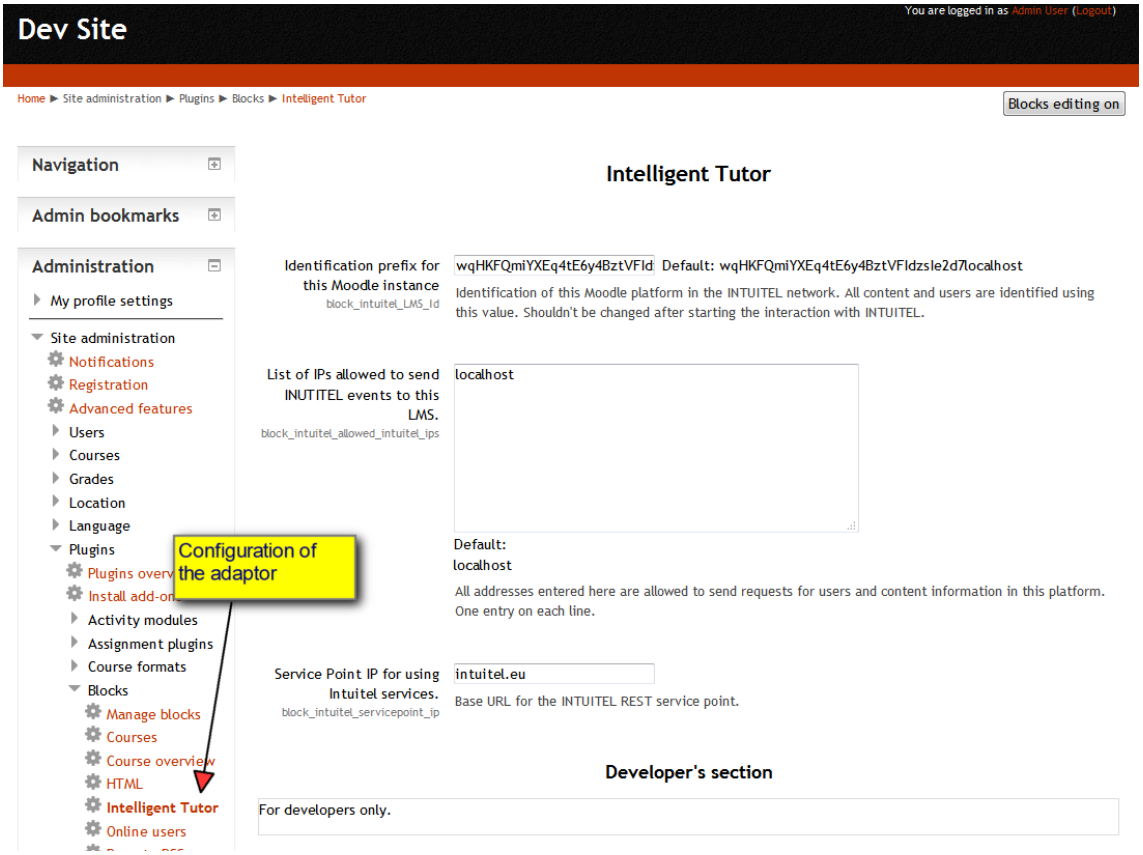

Figure 17: Configuration options of the INTUITEL adaptor

3.2.11.4 How to enable INTUITEL in a Moodle course

INTUITEL in Moodle is presented as a standard Moodle block with name “Intelligent Tutor” that can be added in any course, once it is enabled by the administrator of the site. First of all, the user needs to have enough permission to add the INTUITEL block to the course (capability `block/intuitel:addinstance` for adding the INTUITEL block). By default, the users with role *teacher, editingteacher,* or *manager*; will be able to add the INTUITEL block in the course.

With the edition option activated, the user can activate the INTUITEL interfaces for the course by adding the block “Intelligent Tutor” into the course, as it is done with other blocks of Moodle (see Figure 18). Next step is to check configuration of the block by clicking on the corresponding icon so that the configuration options are displayed (see Figure 19). For the options “Display on page types”, the option “Any page” must be selected so that the block appears in any page of the course. Besides, the region and the priority with which the user wants the INTUITEL block to be shown, can be selected. It is recommended to make the INTUITEL block very visible by situating it in high priority positions so that TUG messages and LORE recommendations are easily noticed by the student.

Last, the user can select if geolocation of the student will be registered or not to be reported to the Intuitel system.

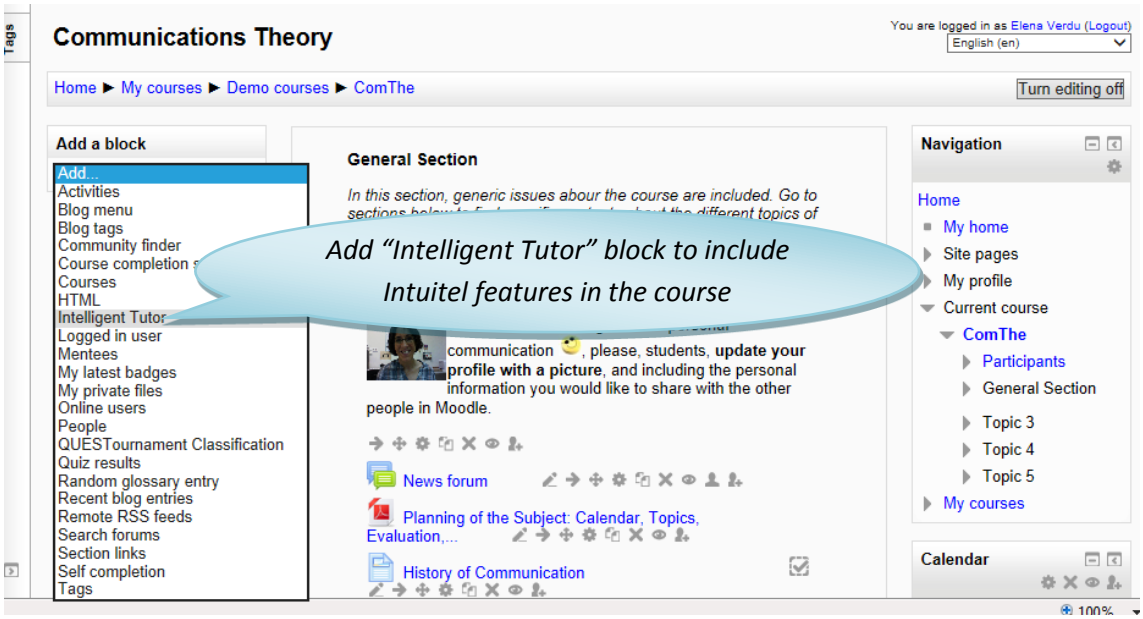

Figure 18: Adding a block through the "add a block" menu in the main screen of a Moodle course

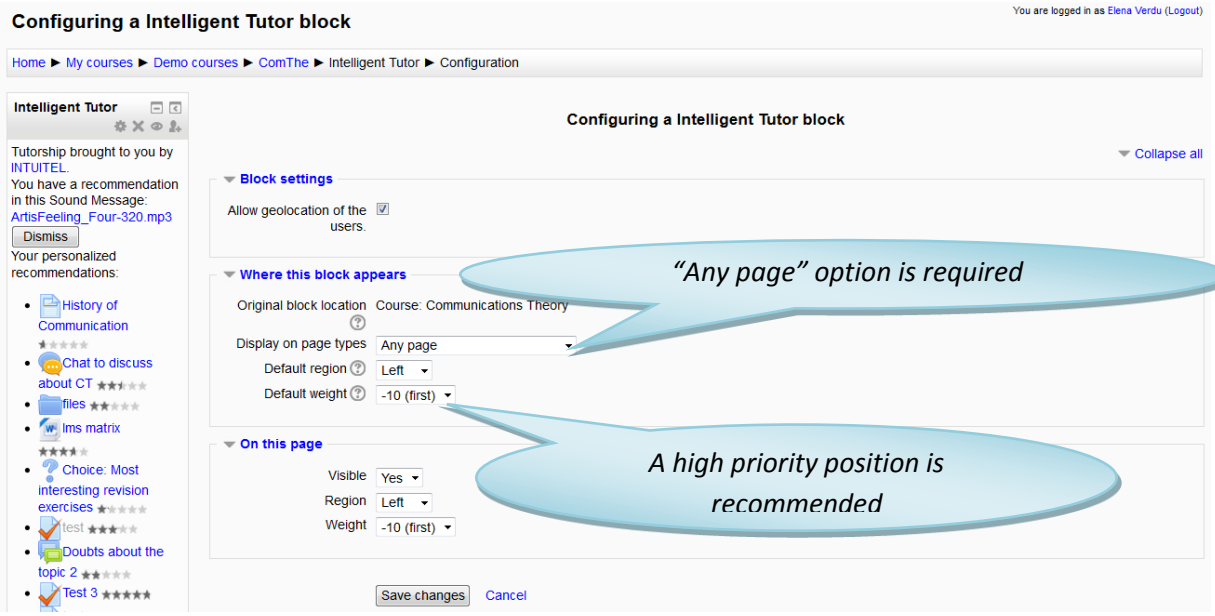

Figure 19: Configuration options of the Intuitel block

3.2.12 Conclusions and Outlook

The current implementation provides a good integration in Moodle in the sense that covers all the specification and generates a non-intrusive user interface that degrades quite well for less capable devices.

Its architecture allows an easy integration path in every existent Moodle installation. It does not make any assumption about server technology or configuration and is directly installable for any Moodle administrator.

This adaptor will not create any kind of technical barrier for installation in small of large Moodle deployments.

3.3 USE/TUG/LORE in Crayons (IOSB)

This chapter describes the integration of INTUITEL in Crayons in terms of architectural aspects, design pattern and implementation guidelines.

To minimize the side-effects with regard to the existing Crayons implementation, the interface-using functions and services are not built into existing source code objects but are rather loosely bound.

This said, the USE/TUG/LORE interfaces is realized as an interface service of its own, only intersecting with existing Crayons code objects where absolutely necessary.

The service implementation is performed in Java (J2EE) with respect to paradigms like REST although not strictly. Like defined by the J2EE standard this service is running on all J2EE-conform application platforms such as Tomcat, Websphere, and so on.

Database access is realized in two different ways. Data belonging to the interface service itself is accessed using standard DAO-patterns. Data belonging to Crayons is accessed using a delegate mechanism making sure the data will not be corrupted. Therefore existing data-access-functions inside Crayons will be reused.

All message driven interface functions are provided for the use in runtime mode. Therefore the client side of Crayons is able to use the interface service as well as the server side of Crayons. This is possible using both: standard URL-encoded, synchronous HTTP-calls (POST and GET) and asynchronous AJAX-calls.

The interface service will not – by itself – distinguish between the two, so the use of a design-by-contract pattern for the remote calls is mandatory.

Figure 20 shows the interface stakeholders - systems and system components, who are authorized to communicate with INTUITEL via the interface.

Figure 20: Interface stakeholders –system and system - components

The stakeholders are:

- INTUITEL system
  - Supports the learner with tutorial guidance (TUG)
  - Presents specific recommendations to the learner (LORE)
  - Determines the learner state and success (USE)
  - Authenticates itself for the LMS
  - Authorizes the LMS
  - Authorizes the runtime environment

- LMS Crayons
  - Provides the author a possibility to includes an "assistance block" with assistance functionalities (tutoring and recommendations)
  - Provides access for the learner to assistance functionality
  - Authenticates itself for the INTUITEL system
  - Authenticates the learner
  - Authenticates the current learning object (and the semantically context)
  - Authorizes the learner

- Crayons Interface Service
  - Communicates with the LMS Crayons
  - Communicates with the INTUITEL system
  - Provides interaction for the learner

3.3.1 Architectural aspects and implementation for Crayons

3.3.1.1 Authoring

Built-in Crayons there is an authoring tool that is used to create Crayons-like learning objects (e.g. courses). To INUITEL-enable a learning object the author is provided with an INTUITEL object - a rectangular area in the clients UI aside the standard learning content. This INTUITEL object can be added easily to an existing learning object.

The use of a block of its own is motivated by the fact that it is easy to update a block's content (asynchronous) without interfering with the rest of the displayed learning content in terms of layout.

The only noticeable change in the generated client code will be in the reaction of learning updates – meaning the user making progress within the learning path.

3.3.1.2 Runtime (Server)

As described above the server provides an interface service dealing with all the communication needs providing USE/TUG/LORE functionalities. In an architectural sense this service can be viewed as a listener and data-broker to both: the client's changes and requests, and the responses and pushes of the assistance services.

During runtime the interface service handles data-requests like a proxy and provides additional functionality like identification (auth), error-handling and logging and such.

3.3.1.3 Runtime (Client)

On the client side the INTUITEL-enabled course-content is enriched with a separate block (DIV) that is used to display all the information that is provided to the learner from the INTUITEL system. To make sure that this fits to the actual learning content – with respect to the learner making progress – all the updates of the learning content is routed through a callback function built-in the INTUITEL block.

Therefore the INTUITEL block (and so the loosely bound INTUITEL systems) knows about the learners status within the course at every single point of time.

3.3.1.4 Implementation

This paragraph describes the basic source code packages and objects that are used to implement the USE/TUG/LORE functionality and the helpers and tools involved.

Java-Packages

Crayons.Intuitel	base package
Crayons.Intuitel.Message	package holding classes for the USE/TUG/LORE related messages
Crayons.Intuitel.Message.TUG	classes to handle TUG messages
Crayons.Intuitel.Message.LORE	classes to handle LORE messages
Crayons.Intuitel.Message.USE	classes to handle USE messages
Crayons.Intuitel.Error	classes and helpers to deal with communication exceptions
Crayons.Intuitel.Logging	logging
Crayons.Intuitel.Proxy	service endpoint during implementation, not needed in production; this endpoint deals with responses to TUG/LORE- requests and performs USE-requests

Crayons.Intuitel.Model	general model – plain old java (data) object classes; These classes represent the data model of the involved data-driven objects (INTUITEL and helpers)
Crayons.Intuitel.Filter	filtering messages (responses) - e.g. filtering out-dated asynchronous responses
Crayons.Intuitel.Service	interface service to deal with all kinds of requests and responses – synchronous and asynchronous; The service deals with all the interoperability functions like setting up the communication, sending and receiving messages, catching communication errors, supervising the message queues and calling the registered callbacks
Crayons.Intuitel.Service.MQ	message queue
Crayons.Intuitel.Dao	[protected] Data objects – all database (in general: persistence) communication goes here. This is the only place in the source code where SQL-statements are allowed.
Crayons.Intuitel.Generator	generators for the client-side INTUITEL-block; extendable to different client scenarios

Java-Libraries

Javax.ws	The Apache Jersey framework is used to deal with REST-related functionality for the interface service. The state transfer is realized in sending learner-ids, message-ids and score-ids (who, what, where/when); the later to easily filter out-dated answers and make the service more 'restfull'.

Javascript-Libraries

crayons.intuitel.client.js	functions for the INTUITEL block on the client side; performs session-handling (cookies), message-handling (AJAX) and – most-important – tracks learner's updates in terms of learning progress
jquery.js	De-facto standard for DOM-manipulation, AJAX-messaging and so on (see jquery.com)

3.3.1.5 Message Life Cycle

Each message (requests and responses) contains a state which consists of several IDs (who, what and where). A message always has two parts: a request and a synchronous or asynchronous response. A

message also acts as a context in which subsequent messages are necessary. For example: A TUG message can (and should) contain a USE message to perform in an efficient way.

The messages are filtered to lower the payload and to filter out unnecessary responses (or requests).

A valid message is stored in a message queue and scheduled for transmission by the service itself. The message queue itself keeps track of the callbacks - if any.

The message life cycle ends with a valid response and the call to a registered callback. In terms of REST no message details are persisted.

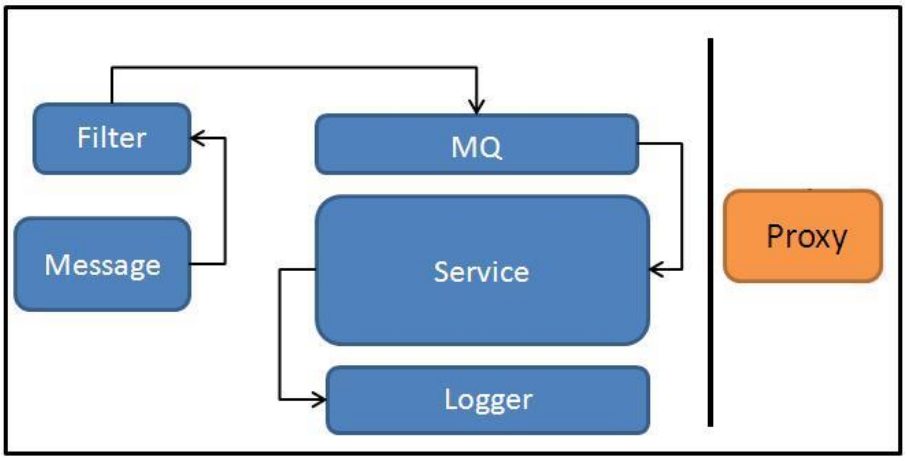

Figure 21: The basic message-driven interface implementation

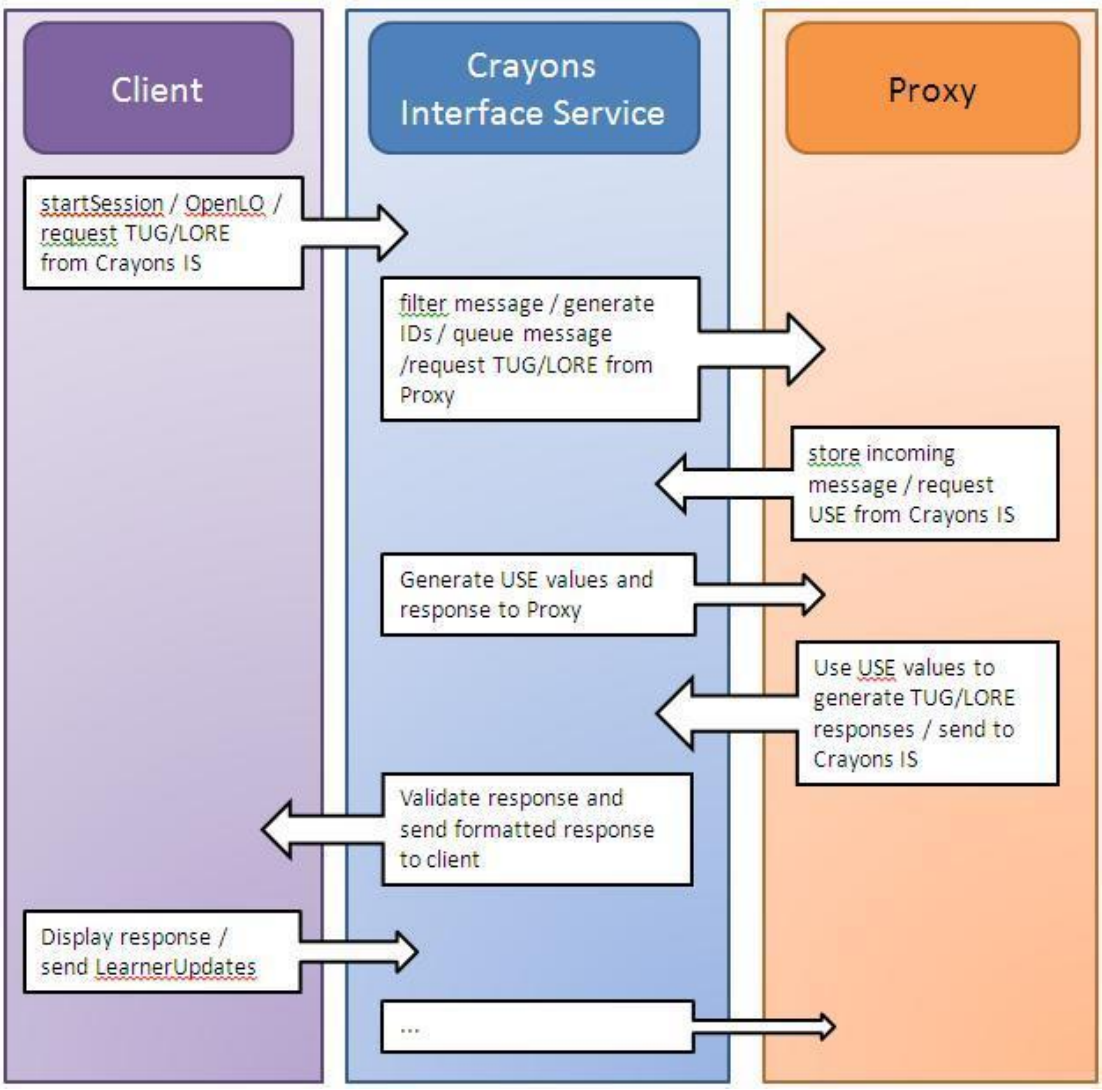

Figure 22: Typical communication scheme

3.3.1.6 Data Model

Based on the existing specifications of data models and communication layer the following data exchange model is implemented:

```
<xs:element name="ROOT">
  <xs:complexType>
    <xs:choice>
      <xs:element ref="LmsProfile" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Learner" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Learners" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="LoMapping" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Authentication" minOccurs="0"
maxOccurs="unbounded"/>
```

```
      <xs:element ref="Tug" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="Lore" minOccurs="0" maxOccurs="unbounded"/>
      <xs:sequence>
        <xs:element ref="UsePerf" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="UseEnv" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

Codelisting 5: data exchange model

Details on the referenced elements (particularly on USE, TUG and LORE) will be found in the respective following chapters.

3.3.2 Lmsprofile

The LMSProfile messages only contain data-elements and a message id. The data-elements are simply a list of named values including the following:

- lmsname The name of the LMS
- lmsversion The version of the LMS
- lmsservice The base path of the interface service
- lmstugpath The path to the TUG service endpoint
- lmslorepath The path to the LORE service endpoint
- lmsusepath The path to the USE service endpoint
- lmscurtime The current time stamp of the LMS
- lmslocale The locale used by the LMS (e.g. ‘us’, ‘de’, etc.)

3.3.3 Learner update

Each interaction of the learner results in a learner update event which is handled by the INTUITEL block. Hence a message is created and a TUG and/or LORE answer is expected.

The filter mechanism of the interface service tracks reported update messages and TUG/LORE answers while the learning object is not changed, making sure that recommendations are only made once during the access-time of that learning object.

The learner update message request is send to INTUITEL every time the user makes progress in the current course, i.e. a new LO is started. TUG and LORE messages are expected as a direct response to that request.

The XML format looks like:

```
<INTUITEL>
   <LearnerUpdate mId="message id" uId="user ID" loId="LO ID"
    time="access time"/>
</INTUITEL>
```

Codelisting 6: learner update xml format

3.3.4 LO mapping and inventory

The LO mapping messages exchange IDs of Learning Objects. The LOMapping function of the interface service provides two-way translation between the INTUITEL-ID and the Crayons-ID of the respective Learning Objects.

Multiple Learning Objects can be 'translated' that way in a single message.

In order to see the mapping of INTUITEL Learning Objects of Learning Objects in the LMS Crayons it is necessary to explain the didactic structure of Crayons. Crayons uses the book metaphor to describe the different levels of didactic.

The first level is the book or more precisely the book cover. It is comparable with the title of a course. A book consists of pages, the second level. Pages can be structured in chapters like we know it from books with subchapters and subsequent chapters, chapters spanning several pages etc. - see Figure 23.

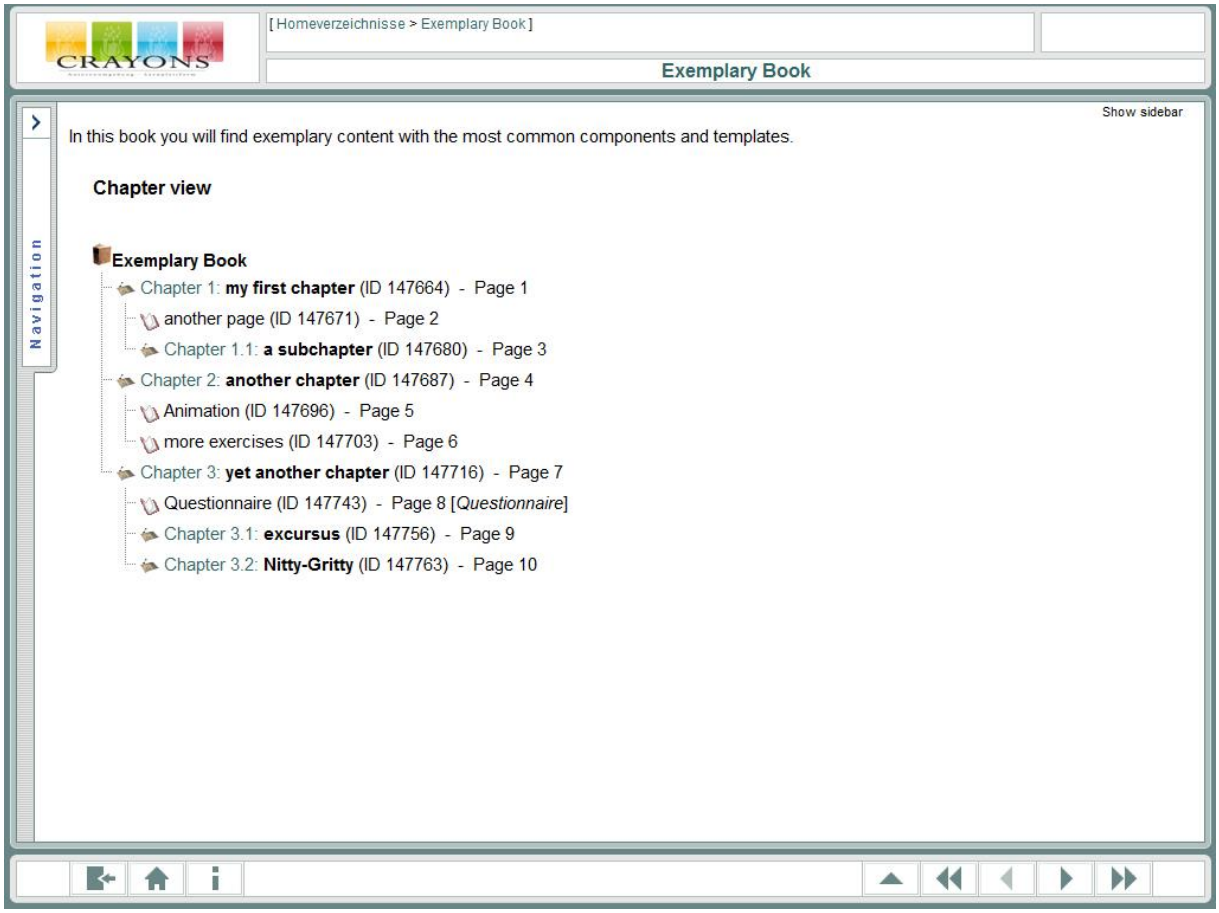

Figure 23: Crayons didactic structure: book, pages, chapters

Yet, pages consist of so called components, the third level. Each media type of a page is a component. There are application-specific text components, media components and exercise components. They are very similar to the media types defined within INTUITEL. Figure 24 shows an

example page. The components of the page are indicated with the red boxes (these are only visible while mouse over in author mode).

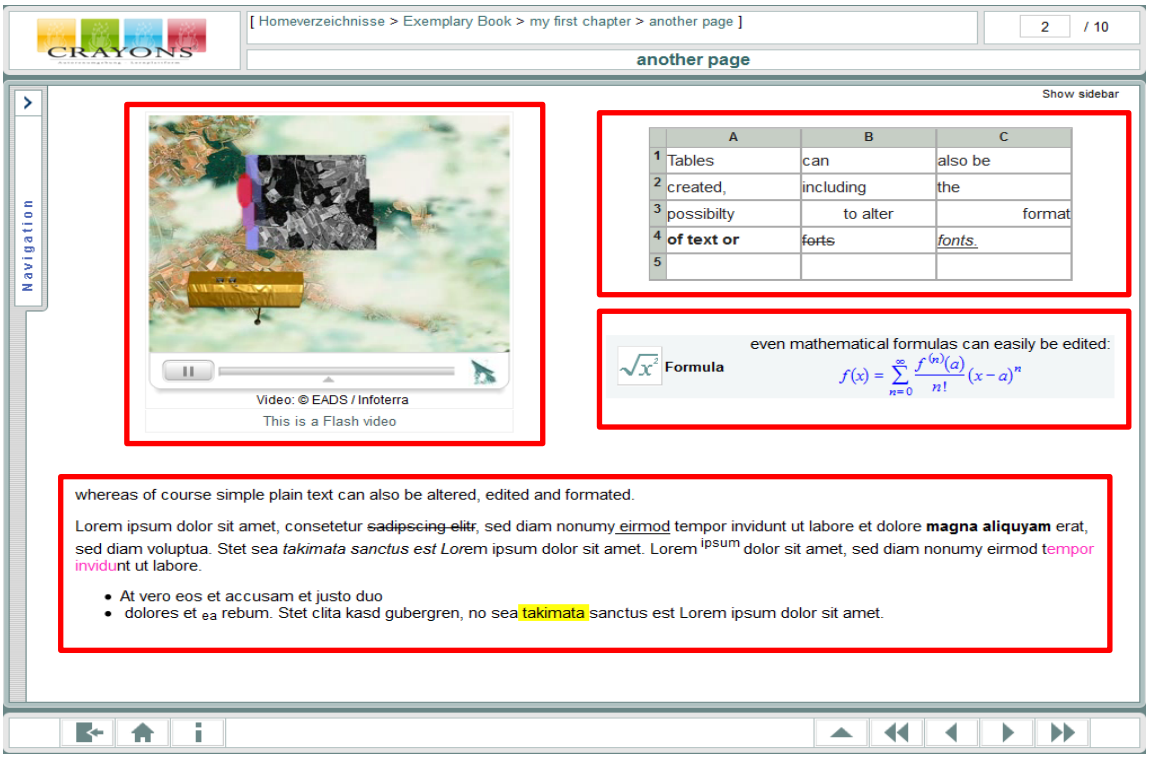

Figure 24: Crayons page components indicated by the red boxes

However based on the definition of LOs in INTUITEL so far it is necessary to map INTUITEL LOs to the level pages of Crayons (see Figure 25). Each new INTUITEL LO will create a new chapter in Crayons containing at least one page. So if the LO is a Concept Container (CC), it is mapped to a new Crayons chapter. Knowledge Objects (KO) will be mapped to Crayons pages. If the LO is both CC and one KO it is mapped to a Crayons chapter spanning one Crayons page.

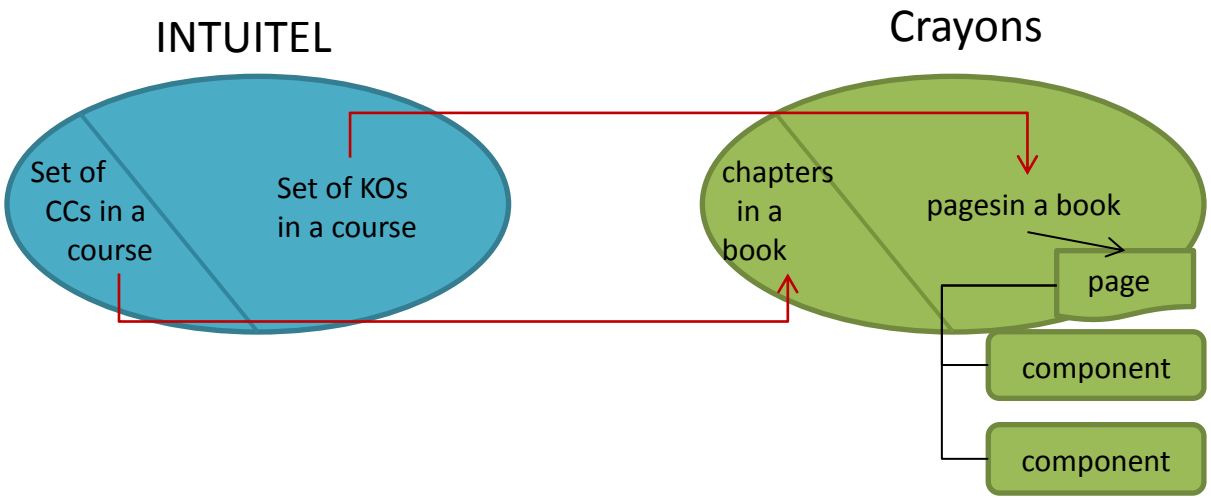

Figure 25: LO mapping

This mapping is convenient for the learner update and identification of LOs in Crayons. The learner update is easily performed and sticks to the Crayons built-in navigation, so for example the UsePerf:seenPercentage can be calculated very simple.

3.3.5 Authentication

Authentication is realized in form of a learner session. During this session a cookie-based session id is used to validate REST request to the interface service. Although this violates the fundamental approach of real REST it eases the way messages can be created and handled (in terms of who/what/where).

Further approaches are still to be discussed, e.g. OAuth2.

3.3.6 Tug

To receive TUG messages, the learner's session has to call for them. While the session is active and the user is making progress, the system expects TUG messages from INTUITEL as a (direct) response to the learner update message.

The content of the TUG message is filtered and provided to the client. There it is displayed within the INTUITEL block.

Possible answers to the TUG messages are:

- Return an error message (ERR) if the user id is unknown
- Return an error message (ERR) if the learning object is unknown
- Return a pause message (PAUSE) if the user has paused the course
- Return an ok message (OK) otherwise

The XML of a single TUG message is as follows:

```
<INTUITEL>
   <Tug uId="user ID" mId="message id" [rId="message/score id"]>
      <MType>message type</MType>
      <MData>message data</MData>
   </Tug>
</INTUITEL>
```

Codelisting 7: single TUG message xml

3.3.7 Lore

As with the TUG messages the LORE messages are expected in the same way: as a result to the learner update message during a TUG/LORE-enabled session.

Possible answers to the LORE messages are:

- Return an error message (ERR) if the user id is unknown
- Return an error message (ERR) if the learning object is unknown
- Return a pause message (PAUSE) if the user has paused the course

- Return an ok message (OK) otherwise

The XML of a single LORE message is as follows:

```
<INTUITEL>
 <Lore uId="user ID" mId="message id" [rId="message/score id"]>
   <LorePrio loId="some LO id" value="integer value in the defined range"/>
         <LoreLoName>name of the LO</LoreLoName>
         <LoreLoDesc>description of the LO</LoreLoDesc>
   </LorePrio>
 </Lore>
</INTUITEL>
```

Codelisting 8: single LORE message xml

The content of the LORE messages is filtered and provided to the client. There it is displayed within the INTUITEL block.

3.3.8 UsePerf

The Use Performance request is asking for a user's score with respect to the different INTUITEL-enabled Learning Objects. The response therefore lists all these scores (according to the defined score types) ordered by the Learning Object ID.

The possible score types are:

- grade (integer)
- completion (integer)
- seenPercentage (float)
- accessed (boolean)

The score type 'internal' is not used.

3.3.9 UseEnv

The Use Environment messages are answered with a list of user-related data. Besides the user id, the following values are delivered:

- fullname the name of the user
- device the device type
- datetime the date and time of the client

3.3.10 Verification programme

A course/book (radar basics, Figure 26) has been created for testing purposes. It contains different Learning Objects and the assistance function augmentation.

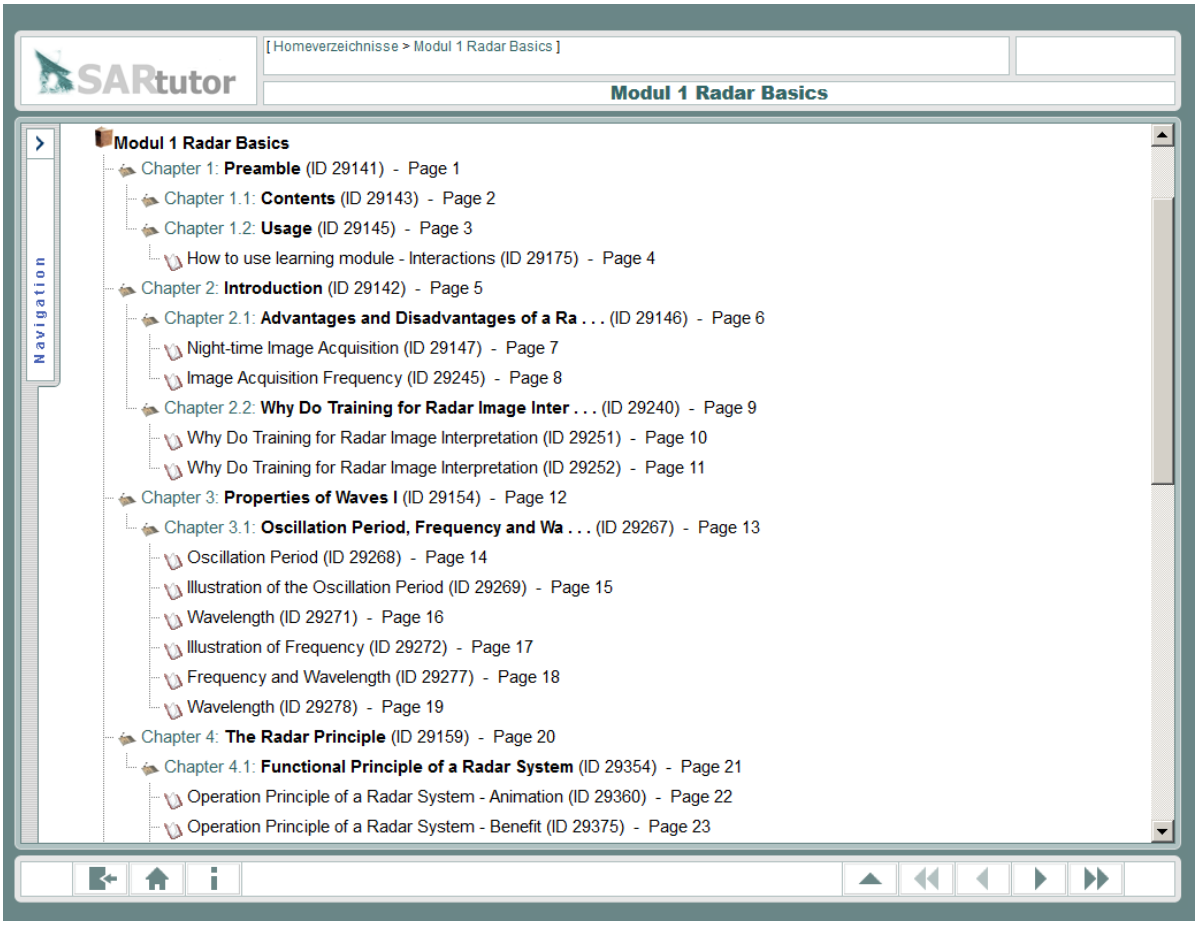

Figure 26: Crayons example course

3.3.10.1 List of test cases

INTUITEL system

- Test Authentication INTUITEL -> LMS.
- Test Authorization INTUITEL -> LMS.

LMS Crayons

- Test integration of assistance functionalities (INTUITEL block).
- Test learner access to assistance functionalities.
- Test Authentication LMS -> INTUITEL

Crayons Interface Service

- Test Communication (changes/requests, pushes/responses) LMS <> INTUITEL

Learner update

- Test if learner's interaction with the course triggers a learner update event.

- Test if the learner update event is handled by the INTUITEL block.
- Test if the learner update message is created.
- Test if the learner update message request is send to INTUITEL every time the user makes progress in the current course.
- Test if filter mechanism of the interface service tracks reported update messages and TUG/LORE answers.

LO mapping and inventory

- Test two-way translation between the INTUITEL-ID and the Crayons-ID of the respective Learning Objects provided by LOMapping function of the interface service.

Authentication

- Test cookie-based session id to validate REST request to the interface service.

Tug / Lore

- Test if TUG / LORE – call is received from the learner's session.
- Test if the call is answered correctly

UsePerf / UseEnv

- Test if the Use Performance request is created.
- Test if the response lists all scores (according to the defined score types) ordered by the Learning Object ID.
- Test if the Use Environment message is created.
- Test if the response returns a list of user-related data.

The implementation is verified to work at the time of submission. Extensive tests as per the use cases in section 3.3.10.1 are planned during October.

3.3.11 Other remarks

3.3.11.1 Discussion on LO mapping

The LO mapping of KOs is done to Crayons pages (see Figure 25). As indicated it would be possible to map them to the deeper level 3, Crayons components. On this level Crayons automatically distinguishes between the different media types as needed by INTUITEL. If we would perform such a mapping to Crayons components, you would not have to choose the main media type of the KO, there would only be one exact media type without having an author to specify it further to the INTUITEL Engine.

However as the components are located somewhere on a Crayons page it is not possible to determine the time a learner needs for completing the KO. This is only possible if there is just one Crayons component on one Crayons page. But then again you wouldn't need a differentiation between Crayons page and component.

Nonetheless, if in further development of INTUITEL it proves more suitable to map Kos on Crayons components, this could easily be realized for Crayons.

3.3.11.2 Discussion on authentication, roles and permissions

As indicated we are currently discussing further approaches on the authentication. This will be finalized by the verification process.

Like in Moodle a role in Crayons is a collection of permissions. This set of permissions is defined for all features within Crayons (they are not valid for the underlying and alternate CMS WebGenesis features). A user is part of a role or a group which is part of a role. You can easily control what a specific user is able to see and/or what he is able to do.

You may define custom roles as well in Crayons but the regular roles fit the INTUITEL needs. These roles are mapped to INTUITEL:

- Student: sees the content and can give feedback for learner update
- Teacher: dynamically controls what students can see from the content/course/LOs. He is not able to change or enhance content.
- Author: all permissions of a teacher but can edit or enhance content/course/LOs.
- Administrator: all permissions of an author but can change and assign accounts, roles and permissions.

3.3.12 Conclusions and Outlook

All INTUITEL specifications for the USE/TUG/LORE interface are covered so far with the actual implementation.

The architecture is minimally invasive for both the LMS and the INTUITEL system. The Crayons interface service acts like a middle tier between INTUITEL and the LMS. So it is easy to integrate in existing and upcoming Crayons installations. Furthermore it can simply be switched off and on within Crayons.

Changes within the communication between INTUITEL and LMS can also be performed without having to touch Crayons source code. The changes – if there are any – will most likely affect only the separated Crayons interface service.

3.4 USE/TUG/LORE in CLIX (IMC)

This chapter describes how we integrate INTUITEL into the Learning Management System (LMS) CLIX. Smartphones and tablets are becoming more common in everyday life. Nowadays mobile devices

have increased computation power, larger screens and highly adapted user interfaces. With the decreasing cost of data connections and the evolution of the data connection standards providing fast and always-on internet connections, mobile learning scenarios became one of the hot topics in e-learning in the last years. Motivated by this observation, we have chosen to implement the INTUITEL recommendation and guidance functionality in a mobile learning frontend for the CLIX LMS designed for Android devices, called *CLIX Mobile App* in the following.

As shown in Figure 27, the overall design consists of four major components:

- the *INTUITEL server* itself (left),
- the *CLIX LMS* (right), whose database is storing information about learning objects,
- a set of *CLIX Mobile App* clients (bottom), used by end users to consume learning content,
- and a *middleware proxy* server (middle), serving as a central communication and coordination hub.

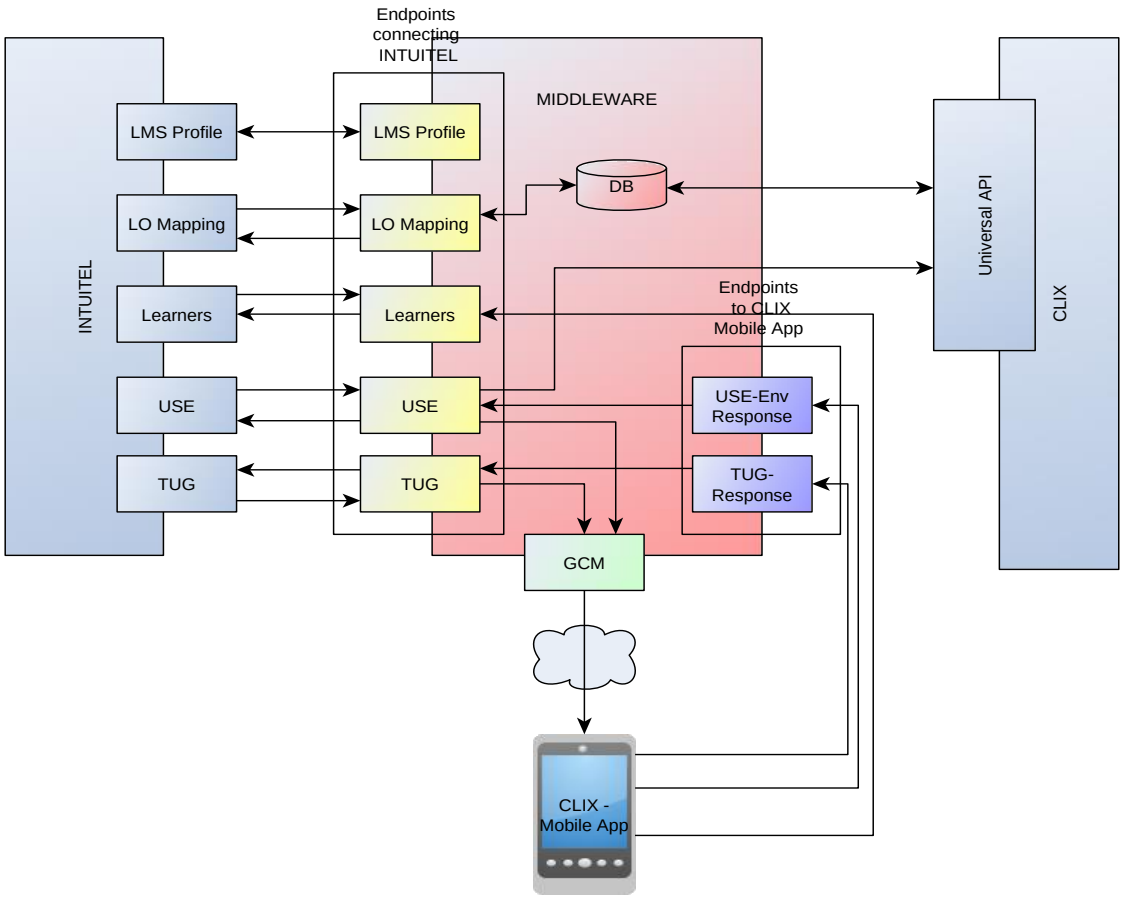

Figure 27: Overview of Middleware Server and Connections

In the following, we describe the work that was done in implementing the middleware proxy server. While the INTUITEL server is not in the focus of this chapter, no implementation changes for the CLIX LMS, which provides an open API for data access, were required. Complementing the middleware

implementation, we will also describe extensions that we made to the CLIX Mobile App, in which we integrated new interfaces for message exchange with the middleware proxy as well as User Interfaces, e.g. for user registration, user login, the display of TUG messages, etc.

As shown in Figure 27, the proxy server sits in the middle of the communication paths and implements routing and translation services. Messages from the INTUITEL server that are supposed to be forwarded to a specific user, i.e. TUG messages, are first sent to the proxy, which then forwards them to the respective CLIX Mobile App clients using the Google Cloud Messaging (GCM) service[8] (we will discuss the GCM service in more detail in Section 3.4.4). In addition to the communication with the mobile devices, the proxy server implements the key application logic for accessing learning objects from the CLIX LMS. To this end, the proxy connects the INTUITEL recommendation system with the CLIX LMS (which maintains a physical database of learning objects) according to the principle of *loose coupling*: access to learning objects stored in the CLIX LMS is realized through a RESTful endpoint, the so-called CLIX *Universal API* (cf. Figure 27), which offers unified access to the learning objects. The benefit of this loosely coupled approach is that no direct code modifications to CLIX are required to access learning objects and, given that the CLIX Universal API is held stable across different release versions of the CLIX LMS, the coupling of INTUITEL does not depend on the CLIX LMS version.

In Figure 27 the implemented midleware and its connections to the other systems are visualised. The enpoints receiving requests from INTUITEL and sending the answers are grouped on the left side of the middleware server. The endpoint providing the profile or configuration data of the LMS parses a local configuration file on requests from INTUITEL and sends the data in XML-format to INTUITEL. INTUITEL requesting a learnig object mapping delivers the result of a query to a local database. The database is updated with data from CLIX.

The learner update to INTUITEL is proactively performed by the mobile device. The mobile device informs the middleware about new user activities and the middleware forwards these updates to the INTUITEL backend.

User Score Extraction messages (USE) are divided in two types of requests, user performance and user environment. For both types a dedicated endpoint is provided. Requests for the user performance are handled by CLIX. These requests received at the user performance endpoint are handed over to the CLIX Universal API after a transformation. The result, e.g. how many points a user achieved in a specific test, is returned. This response is forwarded to INTUITEL by the middleware. In the case of user environment requests, the middleware forwards the requests as a push message to the device of the user, which holds the relevant information. An event handler of the mobile application is invoked and the collected environment data is sent to the USE-Env response endpoint. The middleware then forwads the message to INTUITEL. In case of TUG messages, that is messages and questions which should be shown to the user, the system acts in the same way as for USE environment messages: The message is pushed via Google Cloud Messaging to the device of the user, where the event handler recognizes the TUG message and displays the message to the user. When a

[8] http://developer.android.com/google/gcm/index.html

user decides to answer a message, his/her reply is sent to the TUG-Response endpoint, which then forwards it to INTUITEL.

In the following chapters we will discuss the implementation of the different endpoints and their functionality in more detail.

3.4.1 LMS Profile

The LMS profile information can be accessed by sending a message to the endpoint `<MIDDLEWARE-IP:PORT>/lmsprofile`. An HTTP GET request to this endpoint sent by the INTUITEL server results in parsing a configuration file located on the middleware server. The configuration file is located under `<MIDDLEWARE-PATH>/intuitel/intuitel_conf.js` and contains the data as JavaScript object:

```
var config = {};

/**
 * Installation name of the LMS instance
 */
config.lmsName = 'INTUITEL CLIX';

/**
 * How should the learner be addressed by TUG
 *
 * 0 : neutral,            i.e. "Learner"
 * 1 : forname,     i.e. "Manuel"
 * 2 : surname,            i.e. "Mr. Barroso"
 * 3 : forename & surname, i.e. "Manuel Barroso"
 */
config.INameFormality = '0';

/**
 * Type of LMS instance, here always clix
 */
config.lmsType = 'clix';

 /**
  * Unique ID for identifying LMS instance
  */
config.lmsId = 'THIS_IS_A_UNIQUE_CLIX_ID';

 /**
  * Level of multimedia support.
  *
  * Single character, possible values:
  * v : video
  * a : audio
  */
 config.lmsMediaLevel = '';
```

```
 /**
  * Level of recommendation
  *
  * Possible values:
  * 0 : TUG emulation mode
  * 1 : LORE mode
  */
config.loreLevel = '0';


 /**
 * Level of user score extraction.
 *
 * Possible values:
 * 0 : TUG emulated mode
 * 1 : USE mode
 */
config.useLevel = '1';

 /**
 *
 */
  /**
  * Specifies the method of INTUITEL communication.
  *
  * This is used for learner updates and how INTUITEL
  * Should send LORE and TUG messages.
  * Possible values are:
  * 0 : Pull
  * 1 : Push
  * 2 : Push with learner polling
  */
 config.comStyle = '';
```

Codelisting 9: LMS Profile in the configuration file

The data contained in the configuration file is then parsed and a respective XML-response is generated and sent to the requesting authority (i.e., the INTUITEL server).

3.4.2 Learner update

As for learner update messages, we implemented the push update method for INTUITEL. Technically, the mobile application invokes event handlers whenever a learning object is consumed. The event handler then sends a message to the middleware proxy containing the user credentials, the learning object identifier, as well as a timestamp. This message is then forwarded to the INTUITEL server by the middleware. A new TUG message containing a learning object recommendation (LORE is emulated by TUG interface in our case) is displayed like any other TUG message. For the exchange of

TUG/LORE messages between the proxy server and the CLIX Mobile App clients, the Google Cloud Messaging service is used.

It should be noted that the learner update message currently has two minor restrictions. First, our implementation currently assumes that the mobile device is always connected to the internet when the user is accessing content in the CLIX Mobile app, i.e. messages are sent immediately after a user opened a learning object (and discarded, when no internet connection is available). We are currently exploring different options to queue learner updates in situations where no internet connection is available. Second, depending on the learning object format, the user does not get TUG and LORE messages, e.g. there are limitations when a user accesses a video file or PDFs. The reason is that the CLIX Mobile App opens these file types in an external application, and the lifecycle and security mechanisms of the Android platform do not allow tracking of user activities in third-party-applications. While these problems could technically be solved by a tighter integration of video players and PDF viewers into the learning application, we currently plan to restrict on a prototypical implementation that works for a broad majority of content types and use cases.

3.4.3 LO mapping and inventory

To implement the LO mapping we have to be aware of the following specifics within CLIX: to make learning materials like QTI tests, documents, WBTs accessible to a user in CLIX, the material has to be grouped within a course. The proper user access configuration then assures that a user is only able to see content s/he is allowed to access. More precisely, a user has to be registered to a particular course to interact with its materials; it is not possible that a user accesses learning materials independent of a course. For this conceptual reason, the Universal API allows a specific user only to retrieve learning materials of those courses to which s/he has registered.

The Universal API retrieves the content of a CLIX course by sending an HTTP GET message to `<CLIX-SERVER-IP:PORT>/restapi/lms/courses/<COURSE-ID>/media`. Tackling the user access right problem sketched in the previous paragraph, for collecting the media IDs a special user "INTUITEL" with access rights to all courses (which is created as part of the setup process) is used. The middleware periodically iterates through all courses and fetches the metadata. The metadata of the media types is saved locally in a database. The middleware provides an endpoint at `<MIDLLEWARE-IP:PORT>/mapping` and queries the local database for retrieving the following Learning Object meta-data.

Data Item Name	Description
loName	Object name
loId	Object ID
hasParent	Object identifier of parent
lang	Language code of object

Table 3: LO Meta-Data

3.4.4 Google Cloud Messaging

INTUITEL messages that are forwarded to the mobile device, i.e. TUG messages, the TUG-emulated LORE messages, and the USE environment requests are delivered through the Google Cloud Messaging (GCM) service. Provided by Google for Android devices, this service makes it possible to send data from a server to any Android-powered device, and also to receive messages from these devices over the same connection. The basic idea is that Android applications can register to the Google service to receive push messages (the registration, in our case, is automatically handled by the CLIX Mobile App). Rather than communicating with all the CLIX Mobile running devices directly, messages can then be sent to the Google Cloud Messaging service using the IDs of the registered devices, while the Google Cloud Messaging service takes care of forwarding the messages to the corresponding client devices as soon as they appear online. Hence, no user interaction or periodic pulling is needed to implement message delivery, which greatly facilitates the design of a stable and efficient communication system between our proxy server and the CLIX Mobile App client.

Technically, Google Cloud Messaging is implemented as follows. Once the user logs in into the CLIX Mobile App, the latter registers itself to the Google Cloud Messaging service. As a response, it obtains a unique user token, the client registration ID, which is then communicated to the middleware proxy together with the user name (as discussed in the next section). The middleware proxy maintains a mapping between the users and the client registration IDs. With this information at hand, it can always send messages to users by simply looking up the associated registration IDs.

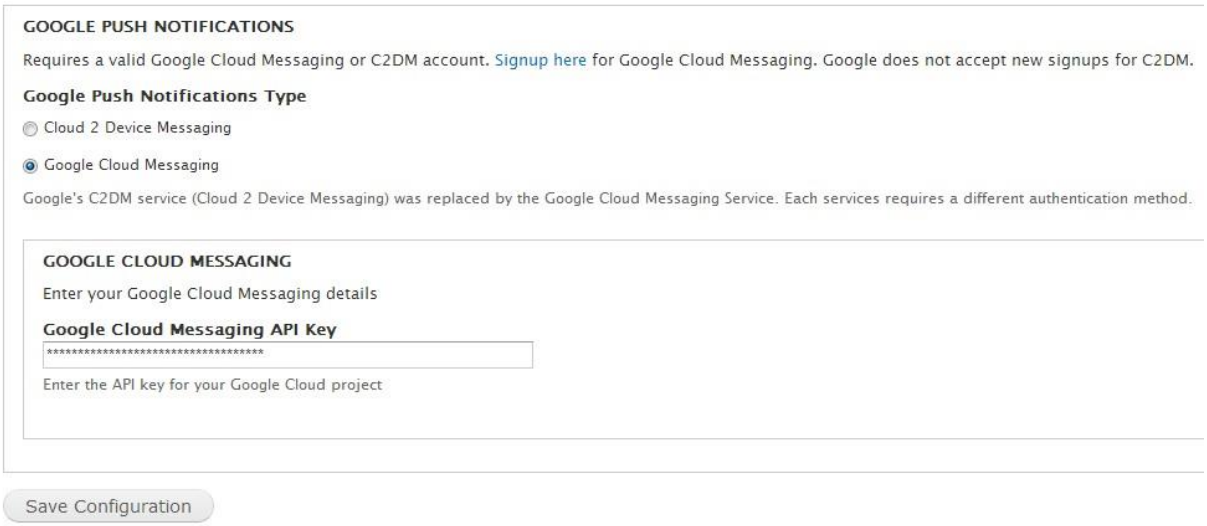

Figure 28: Specifying Google Cloud Messaging API Key at Middleware Administration Frontend

3.4.5 User Device Registration

A new user needs to register a new account via the mobile application. After providing unique login credentials, a new user account is generated at the middleware. To this end, an HTTP POST request is sent to the endpoint `<MIDDLEWARE-IP:PORT>/register`.

```
POST /login HTTP/1.1
Host: localhost
Content-Type: application/json
Accept: application/json
Data:
{
    "name": "A_NEW_LOGIN_NAME",
    "pass": "A_PASSWORD"
    "mail": "A_MAIL@ADDRESS.EU"
}
```

Codelisting 10: Example Registration Request

The payload is in JSON format and contains the name, password and email address specified by the user.

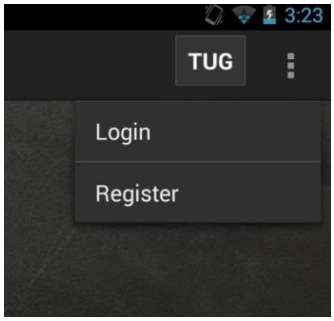

Figure 29: Initiating the Registration Process

The name and the email address must be unique and must not be known before to the system. If a name or an email address is already used, the system replies with a new request for the corresponding data, otherwise a new account is generated and a response confirms the success to the mobile application. After generating a new account, the mobile application automatically registers itself to the Google Cloud Messaging (GCM) service. The unique ID provided by GCM is saved on the mobile application in an SQLite database entry and additionally populated to the middleware server. The middleware server listens on the endpoint `<MIDDLEWARE-IP:PORT>/gcmregistertoken` for an HTTP POST message with the token and the device type, here always Android. The server saves the token and the device type for an authenticated user in a local database. For verification of the user and mapping the token to an account, the Basic Authentication field of the HTTPS message is matched against the user credentials in the database.

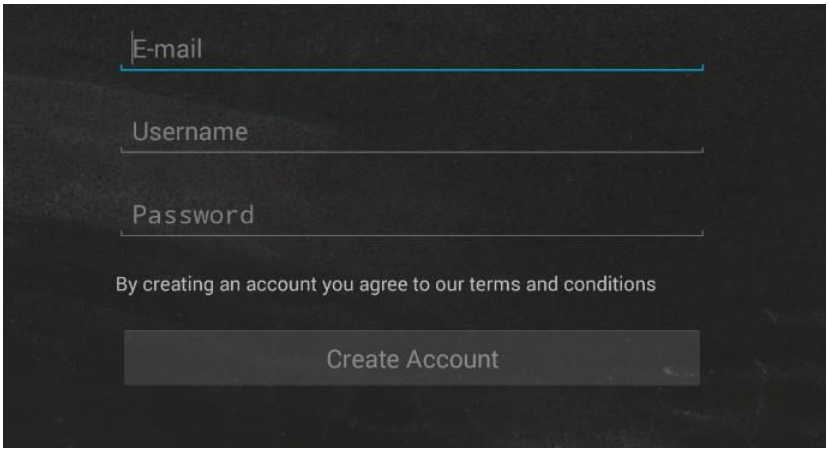

Figure 30: Creating a New User Account

3.4.6 TUG

TUG messages sent to the endpoint at `<MIDDLEWARE-IP:PORT>/tug` are delivered to the user via Google Cloud Messaging service. In a first step the user ID is extracted from the INTUITEL TUG message and the corresponding token for the user is queried from the database. In a next step the message is transformed to JSON format and, together with the token, handed over to the GCM service. The GCM service delivers the message when a user is connected to the internet or saves the message for later delivery.

When the Android device is connected to the internet, GCM automatically delivers the TUG message to the device. An event handler on the device is invoked for treating the event. In the upper right corner of the mobile application a TUG message indicator is located (see Figure 31 for the TUG message button). On delivery of a new TUG message the button is blinking three times and switches its text colour, indicating the user that a new message has arrived. By clicking on the button the message is displayed to the user. Figure 31 shows how the message type “Single Choice Question” could be displayed:

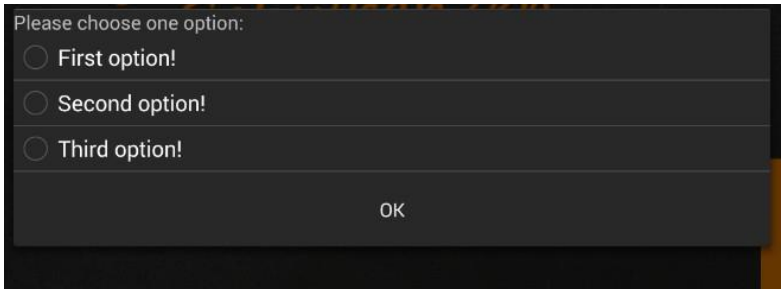

Figure 31: TUG Sample Message with Multiple Options displayed as overlay to the user

Text messages, possibly with basic HTML formatting, can also be displayed. A full specification of the required HTML elements has not yet been given within INTUITEL and extensions may become necessary.

We have chosen the TUG button with overlay display for the following reasons. On the one hand access to TUG messages should be ubiquitously possible within the app – if INTUITEL is enabled for the app, on the other hand the display of the messages should not directly interfere with the already existing thoroughly designed basic design of the app. The button solution also makes it easier to switch between INTUITEL enabled and INTUITEL disabled versions of the CLIX Mobile App.

3.4.7 LORE

The Learning Object Recommender (LORE) is emulated in this mobile scenario via the TUG interface. This means that the learning recommendation is presented to the learner in terms of a TUG message. We did not introduce an extra "LORE button" because wanted to make spare use of additional blocks and buttons, given the limited screen size of mobile devices.

3.4.8 USE Performance

Requests from INTUITEL to collect data, e.g. if a user has visited learning objects or how many questions were correctly answered in a test, are sent to the endpoint at `<MIDDLEWARE-IP:PORT>/USE/performance`. The middleware then redirects the request to the Universal API of CLIX and a callback function is invoked when the result form CLIX is received. The result from CLIX is finally forwarded to the INTUITEL server.

The following score types can be extracted:

Data Item Name	Description
score	How many points a user has achieved in a test.
ratio	The percentage of correct answers a user has given.
status	For WBTs: "WAITING", "STARTED" or "FINISHED" for QTI tests: "PASSED" or "FAILED"

Table 4: Extraction of Score Types

3.4.9 USE Environment

Messages sent to the `<MIDDLEWARE-IP:PORT>/USE/environment` endpoint indicate that the INTUITEL system is requesting environmental data about the user. Analogously to the TUG message delivery process, the corresponding GCM token of the user is queried from the local database. Afterwards a new request is generated. The request contains the mID and the type "USE" in JSON format. Next the request is sent to the user by the GCM service and received by the Android device. The event handler for receiving Google Cloud messages is invoked in the application. The environmental data is then aggregated and a response containing the data and the mID is generated and sent to the middleware endpoint at the `<MIDDLEWARE-IP:PORT>/USE/envresponse`. The JSON data is transformed into XML and sent to INTUITEL.

```
{
    "type": "use",
```

```
    "mId": "1234c678-1234-ab2d-efgh-12c456c89012"
}
```

Codelisting 11: Example USE Environment Request Delivered to an Android Device

The following environmental data is collected and retrieved.

Data Item Name	Description
eTime	Current time of the device
dType	Type of device, i.e. “tablet”
dConType	Type of internet connection, e.g. “umts”, “edge”
dRes	Screen resolution of the device, e.g. “2560x1600”
dBattery	Current battery state in percentage value

Table 5: Extraction of Environmental Data

3.4.10 Verification

In the current phase we focused on manual testing of the implemented functionality. For verifying the correctness of the implementation, the open source program “Postman – REST Client” has been used. Postman is an extension for the Google Chrome browser. It provides an interface for sending HTTP REST requests and visualises the responses. Requests can be stored and organized in collections, making it possible to re-apply a request at a later point in time.

Within Postman, we created sample requests for the different message types and grouped them in collections. In order to verify correctness, we set up the system, populated the system components with sample data, and manually verified that the message responses adhere to the specification. The following screenshot shows an example GET message (sent through Postman) asking for the LMS profile information of a local server instance; the message result, providing basic information about the system configuration, is displayed in the box at the bottom.

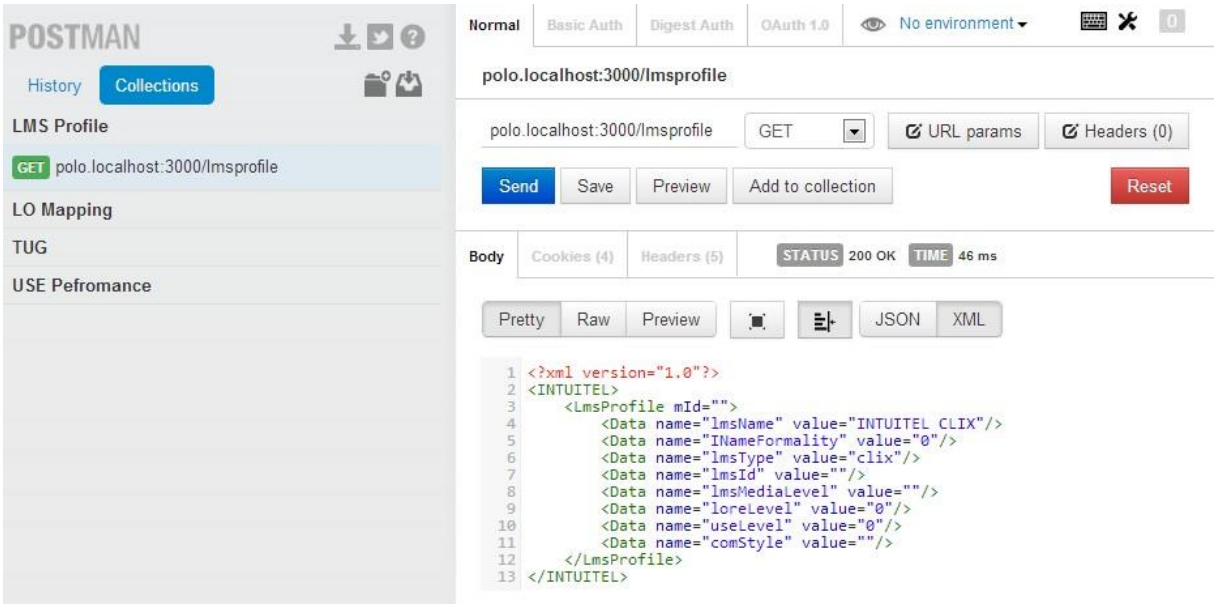

Figure 32: Example GET Message to lmsprofile endpoint with Response

The functionality tests for the API were complemented by a set of manual integration tests, mostly focusing on the CLIX Mobile App. To give an example, these tests verify whether a TUG message sent to a user successfully pops up in the user's mobile device – thus covering more complex workflows involving the proxy server, the Google Cloud Messaging service, as well as the CLIX Mobile App.

In the future, we plan to set up an infrastructure for automated tests based on JUnit. With such a framework, the manual API tests could be fully automated. Moreover, based on automatically generated requests, such an automated test framework makes it possible to implement performance and scalability tests for the middleware proxy, which are hard to achieve by manual tests.

3.4.11 Conclusions and Outlook

In the previous sections we have shown the basic architecture and implementation details of connecting CLIX to the INTUITEL services. The architecture follows the principle of *loose coupling* and is therefore very flexible when it comes to changes both within the professional LMS CLIX and within INTUITEL.

One assumption of the INTUITEL project is that for its recommendation services to properly function a user has to have access to the Internet. For the currently implemented connection to the mobile app we also rely on this assumption being aware that a proper mobile scenario should handle offline situations as well. This includes a proper handling and queuing of messages and events that cannot be directly sent to or received from the INTUITEL server. Some of these questions will be discussed with the INTUITEL team in the next iteration and may be integrated into the mobile app. The results will also influence and improve the current implementation of the Learner update (cf. issues in section 3.4.2).

In the current setting INTUITEL enhanced access to the Learning Management System CLIX is possible through an Android based mobile app. We are currently considering extending the basic architecture

and services to also allow INTUITEL enhanced access to CLIX via a universal HTML based client (also connected to CLIX via its Universal API). Based on a responsive design, this client also implements tailored rendering strategies for different types of devices (in particular Desktop PCs, tablets, and mobile phones), but allows an easier connection to different end-user devices. Furthermore, it would be more straightforward to provide integrated viewers for different types of contents which then makes it easier to track user behaviour when working with the contents (cf. section 3.4.2).

3.5 USE/TUG/LORE in eXact (ELS)

The solution architecture has been implemented in C# using Web API Framework (NET Framework 4.5). This was the choice because the eXact LMS is developed in Microsoft C#.

The communication is HTTP based and the Server side implements a request-response message system, based on XML format, through the implementation of many specific end points each one of which manages only one type of message. The Web Services are developed in C# as a WEB API project; this can give to us more flexibility and more compatibility as a standard .Net Web Service. We choose to use WEB API also because there is a built in support to the XML messages needed in the project.

The solution is a Web Service that waits for a message from the INTUITEL system and then sends an answer. Our solution does not know who sends a message and cannot directly contact external services.

As requested in the Data Model 1.1 no security mechanism is implemented in our solution so the system answer at every correct message from an external system.

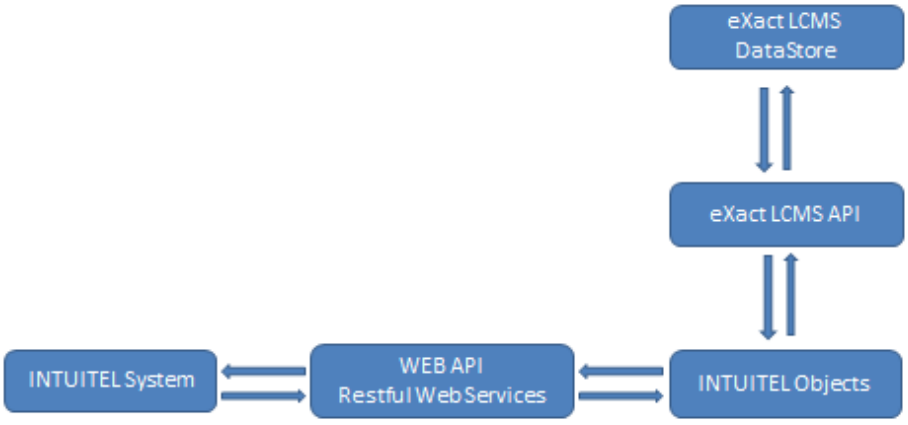

Figure 33: Overview of eXact extension

Intuitel Objects

The Object model has been created using Xsd2Code[9] starting from the supplied XSD as from D1.1. Each object (e.g.: Learners, LMSProfile, etc...) implements two specific methods (e.g.: Serialize, Deserialize) used to marshal and transmit the object representation between the communication endpoints. Also the extractions of the data from the eXact LCMS that are requested from INTUITEL are implemented directly in these objects and not in the Web Services, so we can use the developed object library also with client applications different form INTUITEL and other communication layers.

3.5.1 lmsprofile

We store the data of the "lmsProfile" request in an external configuration file, so we can modify the values for every installation of the LMS. Nothing is stored in the datastore of the LMS or directly coded.

The comStyle attribute is set on "2" due to the "push update" implementation of the learner controller; see section 3.5.2 for more details. Figure 34 represents the "lmsProfile" developed to handle this request.

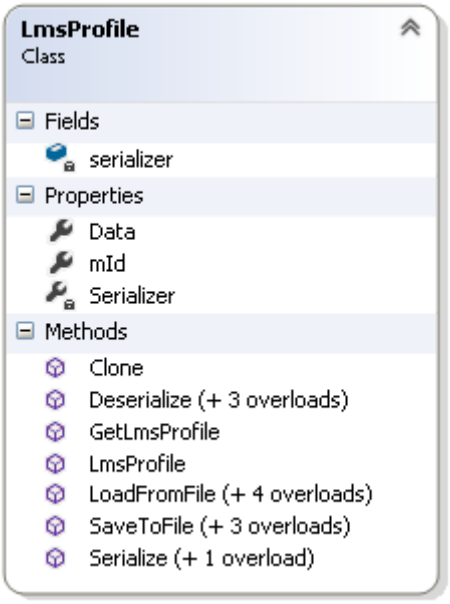

Figure 34: "lmsProfile" object for eXact extension

[9] Xsd2Code is a CSharp or Visual Basic Business Entity class Generator from XSD schema, see http://xsd2code.codeplex.com/

3.5.2 Learner update

We implement this service with the "Push Update" mechanism as described in the D1.1 documentation. We implemented the "learners" method assuming that INTUITEL will periodically make calls to the LMS and the LMS returns a list of active users and their activity, meant as courses accessed by each user since previous "learners" call. Figure 35 shows the objects involved in handling a "learners" request.

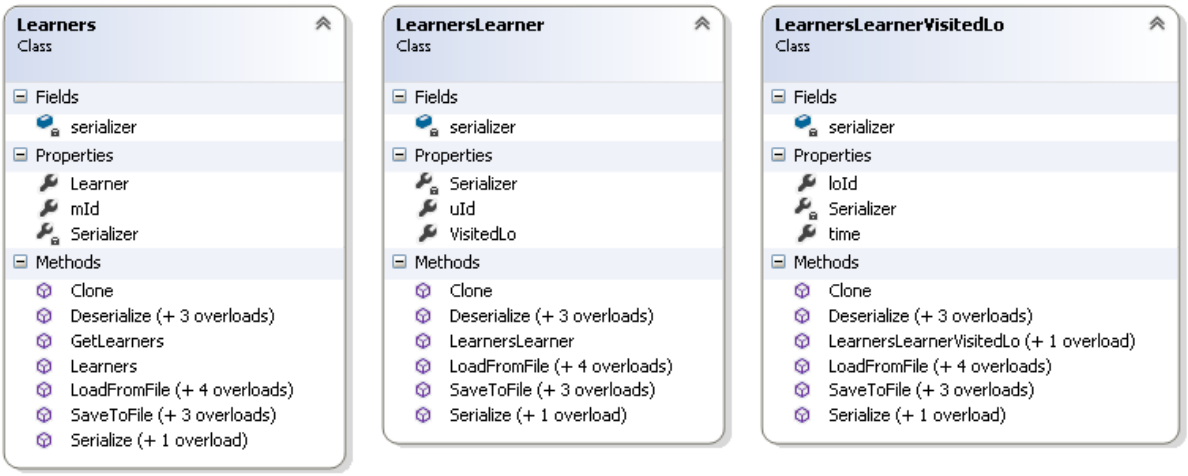

Figure 35: Objects involved in handling "learners" in eXact extension

The object "learners" handles the request from and the answer to INTUITEL request. For each learner into the answer, a "LearnersLearner" object is instantiated, and for each visited LO a LearnersLearnerVisistedLo is instantiated.

3.5.3 LO mapping and inventory

We developed the requested mapping supporting search by title or by LO ID. The corresponding object is represented in Figure 36.

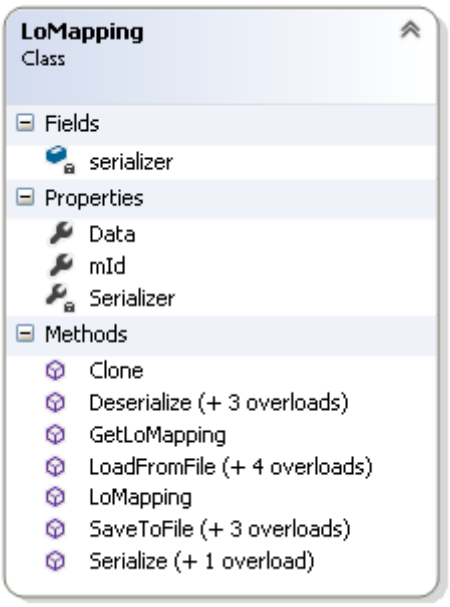

Figure 36: "loMapping" in eXact extension

3.5.4 Authentication

Authentication is implemented as described in D1.1. The LMS receives a request with userId and password and verifies the user's credentials. Then the list of objects over which the user has authorial rights ("edit" right in eXact LCMS) is returned.

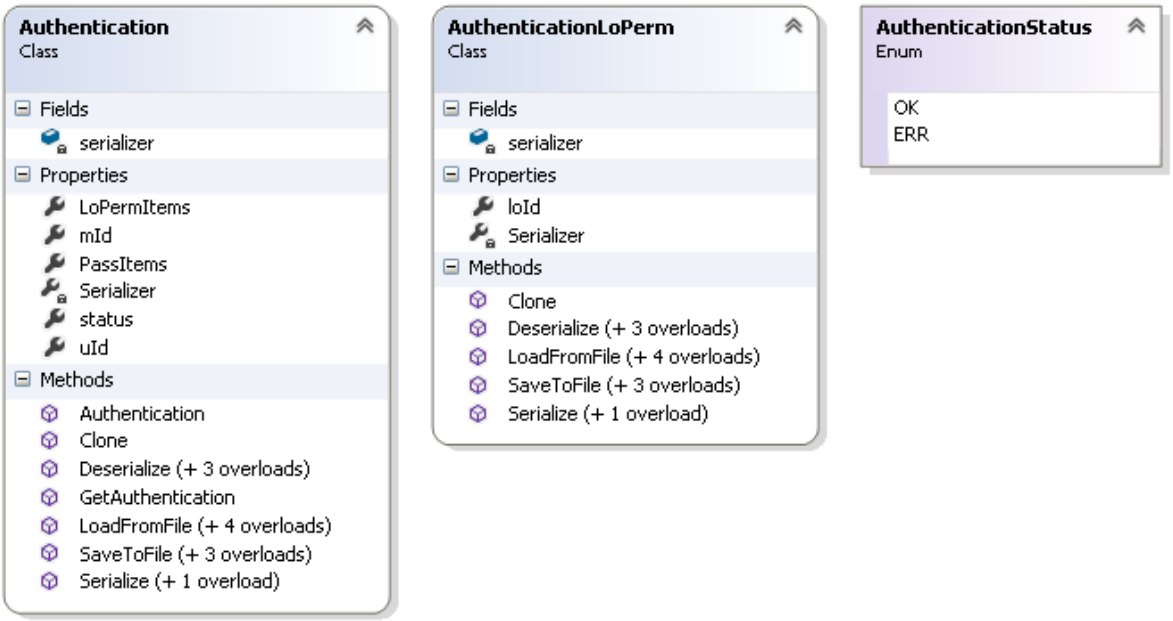

Figure 37: Authentication in eXact extension

Figure 37 shows the objects managing the "authentication" request. "AuthenticationLOPerm" represents the objects id that the (authenticated) user can edit.

3.5.5 Tug

The implemented LMS extension checks if the user, to which the TUG message has to be delivered, is connected to the LMS and then sends back the answer to the INTUITEL system, only with the OK, ERR or PAUSE value. The message is then stored into the LMS datastore.

The TUG messages are directly retrieved by the user interface (web based) from the browser.

We create a new end point in our solution. This endpoint sends the TUG messages to the web client (browser). To retrieve the TUG messages we developed a JavaScript library, this is integrated into the courses web page.

This JavaScript library connect to the Web Services sending the User ID to check if the INTUITEL system sent one or more TUG messages to the user. Then the message is deleted from the LMS datastore.

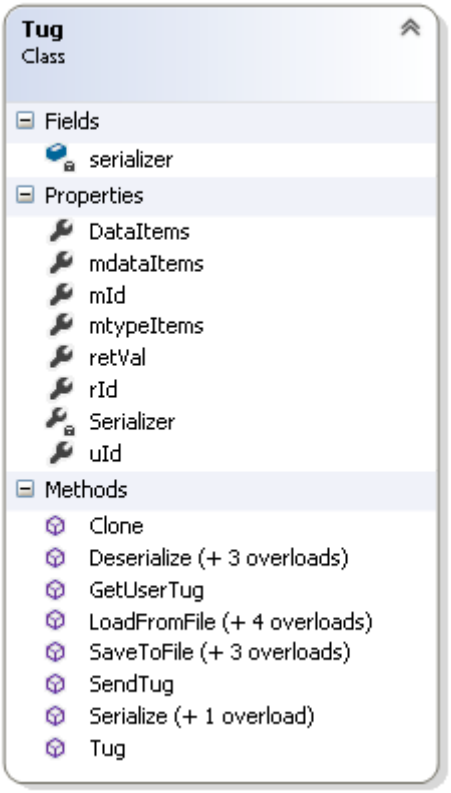

Figure 38: Tug in eXact extension

3.5.6 Lore

INTUITEL recommendations are presented to the user with a layer in the web page. The LORE level is 1.

As described for the TUG messages, the LORE messages are managed in the same way.

1. The extension receives the LORE message
2. The extension checks in the LMS if the user is connected

3. The extension sends back the answer to the INTUITEL system (OK, ERR or PAUSE)
4. The extension stores the LORE message in the LMS datastore

Then we developed a new end point in the Web Service to send back the LORE message to the correct user. This end point is done by a JavaScript library that is used in the Web Client of the LMS:

1. The JavaScript sends a message to the Web Service to ask if new LORE messages are available for the current user, sending the User ID.
2. The extension checks in the LMS datastore if LORE messages are present for the user
3. The extension sends back to the Web Client the messages, if they are present
4. The extension deletes the messages from the LMS datastore
5. The JavaScript library creates a new layer in the Web Client to show the LORE message to the user.

Figure 39 shows the objects handling a LORE message.

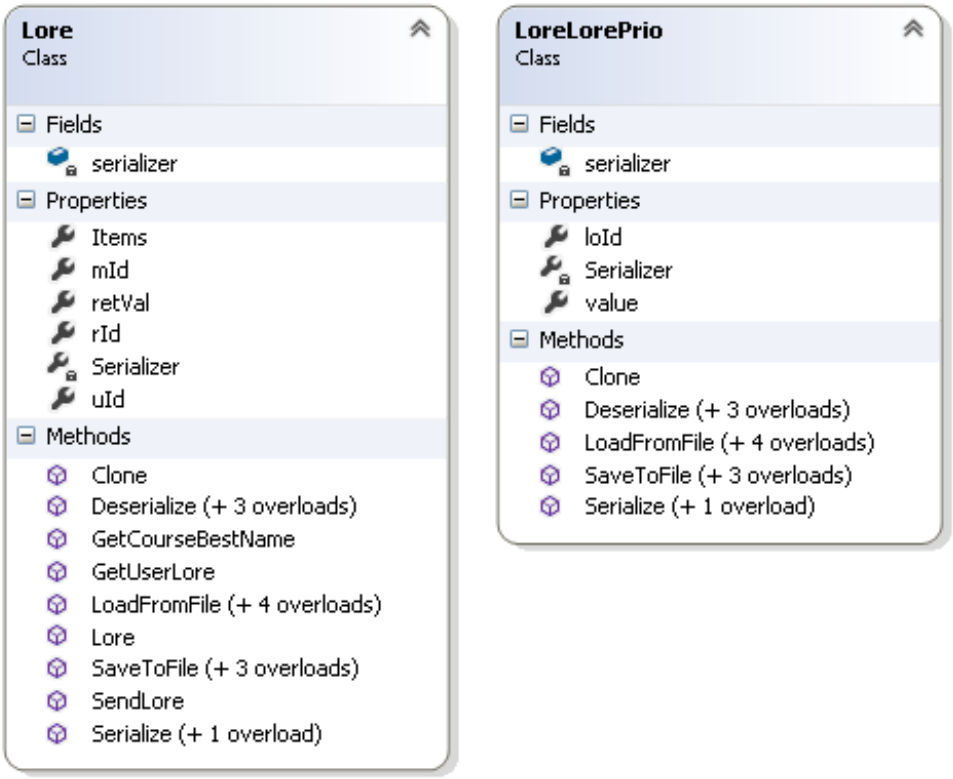

Figure 39: Lore in eXact extension

The “LoreLorePrio” object corresponds to a single recommended LO, while “Lore” represents the whole message.

3.5.7 UsePerf

The information is retrieved directly from the LMS. For each user a list of LOs is provided. For each LO the following data are returned:

- Grade: this corresponds to LO completion status recorded by the LMS, with the following mapping:

- 6 ←→ complete
- 4 ←→ incomplete
- 2 ←→ to start
- 1 ←→ undefined

- Completion: this is what the LMS usually calls “completion percentage” so there is no need to further map such value
- seenPercentage is not supported
- accessed is straighforward

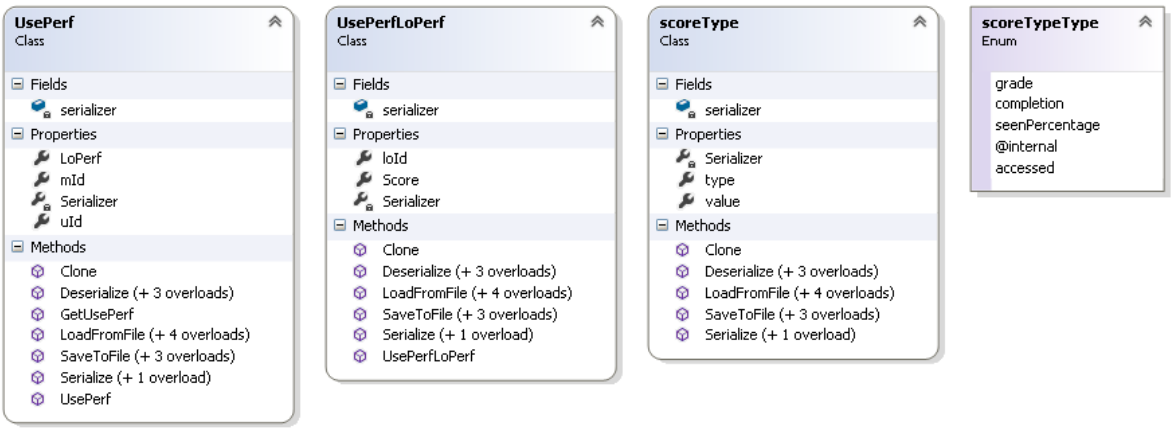

Figure 40: usePerf in eXact extension

3.5.8 UseEnv

As the LMS does not store natively most of the data required for this function, we developed a JavaScript inserted in the WebClient, used to gather the missing information.

Data retrieved from the LMS, if present for the specified user, are: lName, lGender, lAge.

Data newly added and handled by the LMS extension for INTUITEL are: lCulture, lAttitude, eNoiseLvl, eTime, dType, dConType, dConStab, dRes, dBattery.

Some of these data (lCulture, lAttitude, eNoiseLvl, eTime, dType, dConType, dConStab) are asked directly to the user with a survey, each question has multiple choices. The survey is presented to the user but it is not mandatory, so we cannot guarantee that all the fields will be always available to the INTUITEL system. dRes, dBattery are retrieved via JavaScript directly from the user’s browser window.

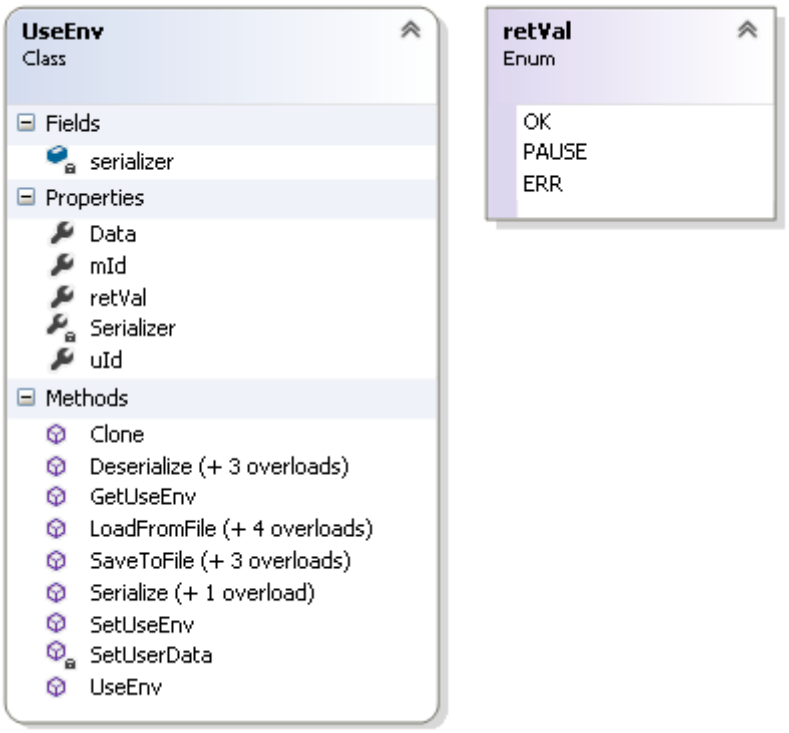

Figure 41: UseEnv in eXact extension

3.5.9 Verification programme

We developed a verification programme as a standalone windows form application (wcf) written in C#. Here we use only the .NET Framework 4.5 methods to call the WEB API End Points. All is modelled using the same INTUITEL Object Class that we use in the WEB API solution.

The verification programme, which interface is presented in Figure 42, instantiates and fills an object of each Type and calls the specific endpoint. As we need to send an entire XML fragment to the WEB API, all the calls are of type POST (see also section 3.5.10.1).

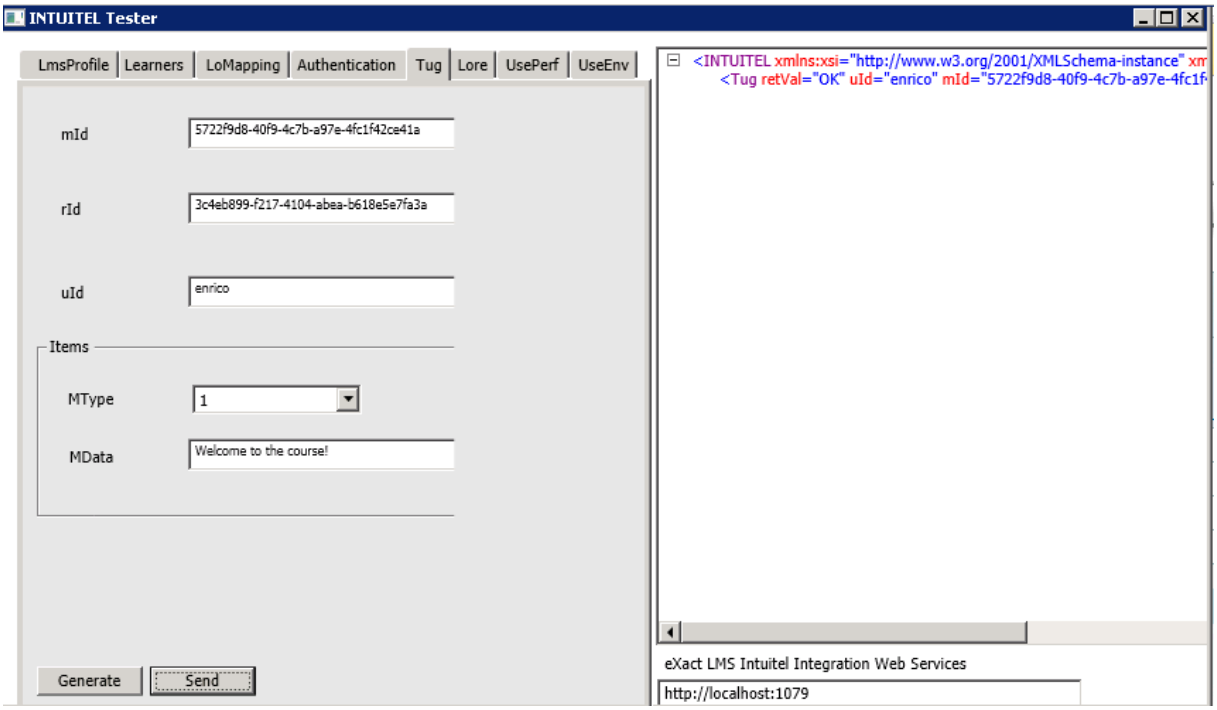

Figure 42: eXact verification programme

Into our verification programme, as from Figure 42, there is one tab for each method of the USE/TUG/LORE interface. For each tab the related parameters are requested for execution. The "Generate" button generates a random message id, while the "Send" button performs the actual call of the method. The right panel shows the xml sent by the LMS extension.

A report of tests and related results are presented in Figure 43. All tests were successfully passed, no errors, bugs nor anomalies were detected.

INTUITEL	*Data Model 1.1*			
eXact LCMS	*eLex_2013_1_4965_90*			
eXact INTUITEL	*1.0.0.0*			
Environment:	*Microsoft Windows Server*			
	Microsoft IIS 7			
	Microsoft SQL Server 2008R2			
	Microsoft .NET Framework			
Test Number	eXact INTUITEL Integration	Short explanation	Result	Status
1	lmsprofile	Send request to the endpoint with auto generated mid	The configuration stored in a xml file on the	OK
2	learners	Send request to the endpoint with auto generated mid	No users connected to the LMS	OK
3	learners	Send request to the endpoint with auto generated mid	Only users without associated LO are	OK
4	learners	Send request to the endpoint with auto generated mid	Every user is returned with the associated	OK
5	lomapping	Send request to the endpoint with auto generated mid without	The whole list of LO is returned	OK
6	lomapping	Send request to the endpoint with auto generated mid with	Only the data of the requested course are	OK
7	lomapping	Send request to the endpoint with auto generated mid with	Only the data of the requested course are	OK
8	authentication	Send wrong user ID	ERR status returned	OK
9	authentication	Send wrong Password	ERR status returned	OK
10	authentication	Send correct user ID and Password of a user without edit	OK status returned	OK
11	authentication	Send correct user ID and Password of a user with edit rights	OK status returned, and a list of LO	OK
12	tug	TUG message with worng user ID	ERR status returned	OK
13	tug	TUG message with user ID not connected	PAUSE status returned	OK
14	tug	TUG message and user connected to the LMS	OK status returned, message stored and	OK
15	lore	LORE message with worng user ID	ERR status returned	OK
16	lore	LORE message with user ID not connected	PAUSE status returned	OK
17	lore	LORE message and user connected to the LMS	OK status returned, message stored and	OK
18	USE/performance	Request with wrong user ID	Only the mid is returned	OK
19	USE/performance	Request with only user ID, no LO id specified	The whole list of LO is returned	OK
20	USE/performance	Request with wrong user ID, a list of LO id is specified	Only the selected LO are returned	OK
21	USE/environment	Request with wrong user ID	ERR status returned	OK
22	USE/environment	Request with user ID not connected to the LMS	PAUSE status returned	OK
23	USE/environment	Request with user ID connected to the LMS	OK status returned, with all availables data	OK

Figure 43: Results of tests for eXact LMS

3.5.10 Other remarks

3.5.10.1 Choice of POST for sending data

The Web Services are all implemented to receive a POST command, the access method is always POST. We use this implementation because we need POST to send an entire XML message to the Web Service. GET instead is usually used for sending (via URL) small data such as strings and not big fragments of XML, as it seems the case here for INTUITEL methods.

3.5.10.2 Handling Mtype and user's answer for TUG

While Mtypes 1 and 2 are quite straightforward to handle, there are some uncertainties about Mtypes 3, 4 and 5. For the moment we assume to display the TUG html message as it arrives. Another pending decision is how and by which formalism the user's answer can be returned back to INTUITEL. We will check further discussions and decision among the consortium.

3.5.10.3 Geolocation

The specification for the useEnv was kept quite open by purpose. Proceeding with the implementation the need of a greater level of detail for Geolocation data emerged and gave origin to a discussion among the consortium.

About the representation of geolocation data we would go for the API specification from W3C[10] using and external library[11]. In this way we would be able to provide the user coordinates, this only if the user accepts to provides his/her location, of course, and his/her browser supports HTML5.

We plan to check if and how to update the implemented extension during the remaining months of WP1 according to the discussion results.

3.5.10.4 GUI

Early ideas and implementation about the GUI are presented in Figure 44 .

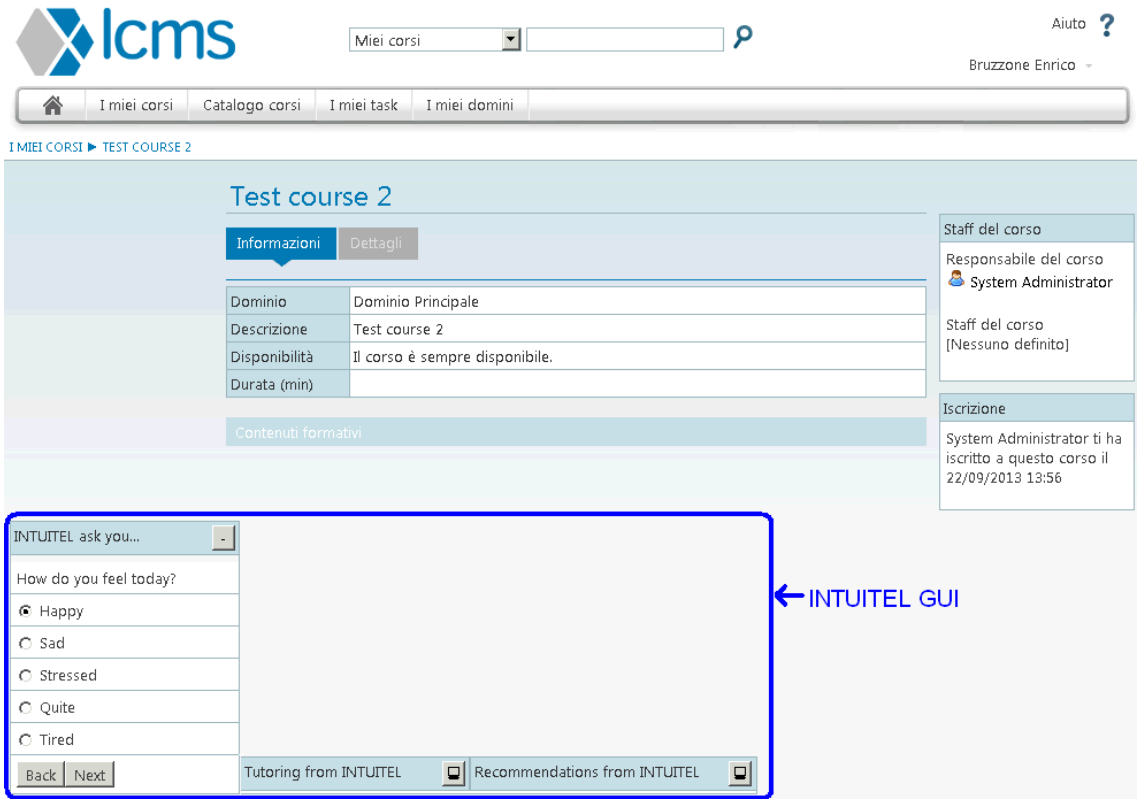

Figure 44: INTUITEL GUI for eXact LMS

As from Figure 44, the choice was to present INTUITEL-related messages on the lower left part. There are three frames, each one can be expanded or minimized independently. The first one presents a kind of survey to collect user's information for the userEnv method. The second and third frames present TUG and LORE messages.

This early implementation is meant for testing and formative assessment and could be refined and updated accordingly.

[10] http://dev.w3.org/geo/api/spec-source.html
[11] http://diveintohtml5.info/geolocation.html

3.5.10.5 LMS extension options

We have considered two possible configurations for the extension of the LMS. The first one takes into account load balancing with multiple front end instances. The second option doesn't take into account multiple front ends.

Option A: multiple front ends, load balancing

A typical LMS deploy foresees multiple front end instances for load balancing. The extension would look like in Figure 45.

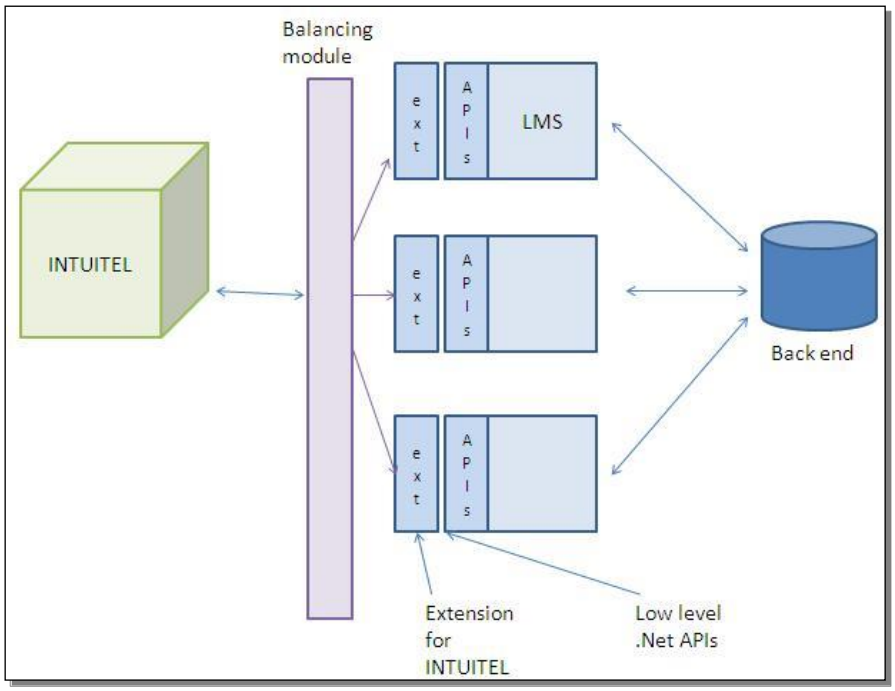

Figure 45: Hypothesis of INTUITEL extension for load balancing deploy.

Option B

A simpler deploy doesn't take into account load balancing and assumes to work with only one front end, as from Figure 46.

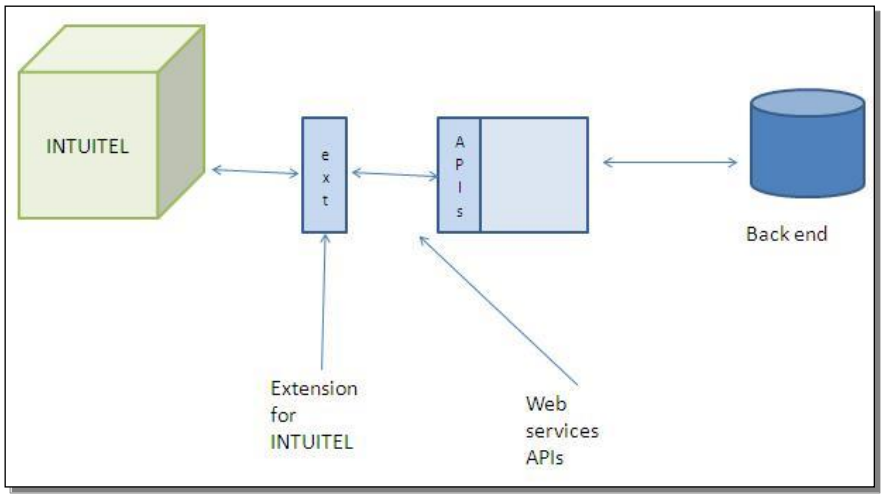

Figure 46: Hypothesis of INTUITEL extension for single front end.

Analysis and choice

A first idea for the extension could be to proceed with the apparently simpler option, which is the deployment with a single front end, for a prototypical approach, with the objective of adding a load balancing manager module in a second time.

But the different deploy modality becomes relevant when considering the LMS answer to certain kinds of requests from INTUITEL and how to process and manage such kind of requests.

For example, in the case of a TUG message, with the option of a single front end it is easily managed in a straightforward manner, via direct answer, while in the case of multiple front ends the message need to be store into the back end and accessed from there by the individual front ends. In this case the later adoption of a custom load balancer would have a strong impact in code update.

As the **load balancing (option A)** is the usual approach for professional, real life deploy we chose this option as it would allow also a quicker adoption and exploitability of the solution.

3.5.11 Conclusions and Outlook

The current specification for the USE/TUG/LORE interface was implemented with our extension. In particular, the classes handling the logic of the methods and the interfaces with the LMs have been separated by the classes managing the communication with INTUITEL. This will allow an easier switch to other/extended communication mechanism, such as the advanced scenario in D3.3.

There are still some uncertainties about some data types, as from section 3.5.10.2 and section 3.5.10.3, and further work will be needed in case the advanced communication scenario needs to be implemented, as this is also related to in-progress discussions and work in other work packages.

4 Overview and next steps

As described in previous sections, extensions for all LMSs have been developed to expose the USE/TUG/LORE as described in D1.1. At the time of submission (September 30,2013), the functionality of the five different interfaces has been asserted and verified.

Tests have been carried on or are in progress until the end of WP01 (end of month 14, e.g. November, 2013). Additional extensive integration testing is also envisaged within WP3 with Tasks 3.3 and 3.4 and within WP12.

- For the Open Source LMS, an automatic testing suite is available (description see section 2.3). At the time of submission of the present paper, the majority of these test gives an expected result and 2-3 (out of 75) show less than sufficient results.
- For the three commercial LMS Clix, eXact and Crayons, test cases carried out manually and screen shots have been documented here, consequently the proper verification has been carried out.
- The consortium expects that live hands-on testing for all five LMS will be available at the scheduled review meeting for INTUITEL (Dec. 10, 2013).

Note in this context also, that the Tasks 1.2 – 1.6 are still running until the end of November, 2013. **The project milestone M2, e.g. the availability of the USE/TUG/LORE interfaces therefore is considered to be achieved as planned.**

As highlighted by comments in previous sections there are also a few open points which arose during development:

- A first challenge is the refinement and/or agreement for the message format with Mtype 3, 4 and 5 of the TUG messages. The consortium expects this to be clarified during development of the INTUITEL content for the four different knowledge domains.
- A second challenge is the representation of Geolocation information. At the time of writing the discussion converged on a format to be used. LMSs wishing to support Geolocation features as well are envisaged to further update their related extension during upcoming months.