Deliverable 5.2

INTUITEL Back End Documentation

DISSEMINATION LEVEL		
PU	Public	**X**
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)	

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.:	D5.2
Document name:	INTUITEL Back End Documentation
Work Package	WP 5
Associated Task	5.1, 5.2, 5.3, 5.4, 5.5, 5.6
Nature [1]	P
Dissemination Level [2]	PU
Version:	1.0
Actual Submission Date:	2014-09-30
Contractual Submission Date	2014-09-30
Editor: Institution: E-mail:	Luis de la Fuente Valentín Universidad Internacional de la Rioja, (UNIR) luis.delafuente@unir.net

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	2014-05-01	Document drafted	Andrea Zielinski	IOSB
0.02	2014-06-10	Document revised	Andrea Zielinski, Luis de la Fuente	IOSB UNIR
0.03	2014-07-01	Document revised	Andrea Zielinski, Luis de la Fuente	IOSB UNIR
0.04	2014-07-10	Structure Finalized	Andrea Zielinski	IOSB
0.05	2014-08-10	Document revised	Andrea Zielinski	IOSB
0.06	2014-09-10	Document revised	Andrea Zielinski	IOSB
0.07	2014-09-29	Document revised	Luis de la Fuente	UNIR
0.9	2014-09-29	Document updated	Andrea Zielinski	IOSB
1.0	2014-09-30	Document finalized and approved	Peter Henning	HSKA

Distribution List

Date	Issue	Group
2014-05-31	First draft submitted for discussion	WP05 leader
2014-06-12	Request for comments on document structure	WP05
2014-07-01	Request for contributions	WP05

List of Contributions

Date	Organization(s)	Person(s)	Contribution
2014-05-01	IOSB	Andrea Zielinski	Initial document structure
2014-06-10	UNIR	Luis de la Fuente	Revised Initial document structure
2014-06-12	UNIR, IOSB	Luis de la Fuente, Andrea	Iterative reviews on the initial

		Zielinski	structure
2014-06-16	HSKA	Luis de la Fuente, Andrea Zielinski	Improvements on the initial structure
2014-06-24	URE	Ovidiu Serban	Improvements on the initial structure
2014-06-25	LAT	Elisabetta Parodi	Improvements on the initial structure
2014-06-27	UVA	Elena Verdú	Improvements on the initial structure
2014-07-10	UNIR, IOSB	Luis de la Fuente, Andrea Zielinski	Initial structure fixed
2014-08-15	HSKA	Florian Heberle	Contributions to sections 3,5 and 7
2014-08-15	URE	Ovidiu Serban Daniel Thiemert, Atta Badii	Contributions to sections 3 and 5
2014-08-17	IOSB	Andrea Zielinski	Contributions to sections 2 and 3.1.2
2014-08-27	HSKA	Florian Heberle	Content improvement
2014-09-01	URE	Ovidiu Serban Daniel Thiemert, Atta Badii	Contributions on section 3.2, 3.4, 5.7
2014-09-01	LAT	Elisabetta Parodi	Improvements on 5.1
2014-09-09	FZI	Jürgen Bock	Contributions on 3.1.3 and 3.3
2014-09-11	IOSB	Andreas Bischoff, Andrea Zielinski	Contributions on 3.4.3.4
2014-09-15	IOSB	Andrea Zielinski	Contributions on 3.4.2
2014-09-15	UVA	Juan Pablo de Castro, Elena Verdú	Contributions on 4.1
2014-09-19	TIE	Oscar Garcia Perales, Oscar Soriano Enguídanos, José Rafael Navalón Baixeras	Contributions on 3.2.1, 5.2 and 5.4, and appendix 9.1
2014-09-12	FZI	Matthias Stumpp, Jürgen Bock	Contributions to 5.6 (Query Builder)
2014-09-19	HSKA	Florian Heberle	Proofreading
2014-09-23	FZI	Jürgen Bock	Contributions to 4.2, 5.6 and 6
2014-09-23	IOSB	Andrea Zielinski	Contributions to 4 and 5.7
2014-09-23	URE	Ovidiu Serban	Contributions to 4.4

2014-09-26	FZI	Jürgen Bock	Issues fixed along the document
2014-09-29	IOSB	Andrea Zielinski	Overall review and conclusions section
2014-09-29	UNIR	Luis de la Fuente Valentín	Overall review and glossary added

Executive Summary

Abstract	One major contribution of this deliverable is the description of the design and implementation of the INTUITEL approach to adaptive learning object selection. The basic task of the INTUITEL ENGINE is to create learner specific recommendations and forward them to the Learning Management System.This approach integrates • Reasoning over the learner state ontology (which includes learner model, learning objects model and pedagogical model) • Ranking learning objects • Integration of feedback messages The main goal of this deliverable is to give an overall introduction to the individual components of the INTUITEL Engine components and show how they are integrated and fit into the overall INTUITEL architecture. A manual report is provided that offers guidelines to install and deploy the INTUITEL system from a developer's perspective [DOW p.22]. The document is structured as follows: Chapter 2 gives the motivation and challenges for an ontology-based framework for adaptive Technology Enhanced Learning. Chapter 3 describes the INTUITEL Engine on a conceptual level, i.e. the Query Builder, the Reasoning Broker, and the Recommendation Rewriter that consists of a Ranking Module and a Natural Language Unit. A new approach to learning pathways as structured sequences is offered, which extends the former Pedagogical Ontology. Moreover, a new methodology for integrating ranking into the reasoning process is described. Ranking takes place in accordance with user-defined weights for the importance of Didactical Factors and the matching similarity between learner profile and learning objects. Chapter 4 is devoted to explain the testing procedures used to evaluate and validate the functionality of the INTUITEL Engine. Chapter 5 offers guidelines to install and deploy the INTUITEL system. Then, Chapter 6 discusses the integration of different reasoning engines in a reasoning broker framework, and finally, Chapter 7 highlights the major conclusions.

Key Aspects	According to the DoW, the INTUITEL reasoning engine should • "by automated reasoning deduce optimal guidance and feedback to an individual user • the deductive process my include the current learner performance (INTUITEL monitors the learner's progress and behavior p.6), metadata of user, contextual data" [p.6] • It draws conclusions from the Learning Progress Model (LPM) based on measurable and measured results [p. 16] and LMS using the current position of a learner in a cognitive model, direction and velocity of learner, metadata provided in the course material, and historical user data [p.14] • Match the user's abilities with the course material. Test results are extracted by a User Score Extractor (USE). • Suggest a next topic and prioritize learning objects [p.8]; the Learning Object Recommender (LORE) will then carry a recommendation factor [p.12] INTUITEL should eventually be used as a recommender system and generate metadata from context [p. 23] • Carry out a dialogue with user (advanced dialogue techniques). Tutor Guidance (TUG) should send meaningful messages [p.10].
Keywords	OWL reasoning, OWL sequencing, ranking, feedback messages, learning object recommendation, INTUITEL User and Developer's Guide

Table of Contents

List of figures

List of tables

1 Glossary

Term	Explanation
Concept Container (CC)	Type of Learning Object on the lesson level. A CC is part of a Knowledge Domain (KD) and contains multiple Knowledge Objects (KOs) to form a lesson.
Cognitive Map (CM)	Description of a domain of knowledge using the terms and relations as specified in the Pedagogical Ontology.
Cognitive Concept Map (CMM)	A Cognitive Content Map (CCM) is the INTUITEL description of an eLearning course using the terms and relations as specified in the Pedagogical Ontology
Communication Layer (CL)	The INTUITEL Communication Layer (CL) is the cornerstone for the messages exchange between all components of the INTUITEL System
Content Creator	A person, usually some kind of lecturer, a domain or ontology specialist, who fulfils the task to create learning material, which can, when completing the whole INTUITEL specific workflow, be used in the recommendation creation process.
Didactical Factor (DF)	Nominal (i.e. non-numeric) value that is composed of a number of data items that are relevant for INTUITEL in order to create an assertion about the learning situation, habits or preferences of a particular learner and, therefore, to recommend suitable KOs and to generate learner feedback.
Knowledge Domain (KD)	Topmost cognitive container of an INTUITEL enhanced 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.
Knowledge Object (KO)	In INTUITEL a Knowledge Object (KO) is an item of knowledge, which typically corresponds to one screen page of content and to an estimated learning time of 3-10 minutes for the average learner.
Knowledge Type (KT)	Categorization of Knowledge Objects according to the type of knowledge that is contained, i.e. their function within the learning process. Different Knowledge Types are defined for different learning approaches: multi-stage-good-practice, multi-stage-simulation, open-inquiry-based and structured-inquiry-based.
Learner	A learning person using the INTUITEL enhanced LMS.
Learner Model Ontology (LMO)	The Learner Model Ontology defines classes and properties for describing the current learner state (see D3.2).
Learning Management System (LMS)	Software application with which content creators, lectures or learners interact. It provides mechanisms to administrate, document, track, report and deliver the educational content.
Learning Pathway (LP)	“Map” describing how to find a suitable route to get the objectives of a course.
Learner Progress Model (LPM)	Central mediating component of the INTUITEL system which acts as a data transformation and distribution entity.

Learner State Ontology (LSO)	The Learner State Ontology contains learner-specific data as instances of the Learner Model Ontology.
Learning Object (LO)	Learning Object (LO) is in INTUITEL the umbrella term for the various types of learning containers, such as Knowledge Domain (KD), Concept Container (CC) and Knowledge Object (KO).
Learning Object Recommender (LORE)	Interface provided by the LMS to show the learner situational relevant recommendations computed by INTUITEL.
Media Type (MT)	Categorization of Knowledge Objects according to the type of contained media.
Natural Language Unit (NLU)	The Natural Language Unit creates the text messages for TUG, based on Reflexes and Didactic Factors, and the RM uses the Didactic Factor data to prioritize the reasoning results for LORE messages.
Web Ontology language (OWL)	A W3C Standard since 2004 that is widely used within the Web of Data for the description of ontologies and which is based on RDF.
Pedagogical Ontology (PO)	OWL-ontology, which provides terms and relations to model courses and learning pathways in a semantically rich way, in order to enable the INTUITEL Back End to automatically create learning recommendations.
Query Builder (QB)	The INTUITEL Query Builder is a component within the INTUITEL Engine that computes sets of KOs as recommendation candidates. Each set represents a selection of KOs according to a particular selection criterion.
Ranking Module (RM)	The INTUITEL Ranking Module is a component that builds on the results provided by the INTUITEL Query Builder and the LPM to retrieve the final result, i.e. a personalized recommendation of suitable learning objects.
Reasoning Broker (RB)	The Reasoning Broker takes care of the selection of suitable reasoners and parallelization of reasoning tasks to optimize the system's performance
Recommendation Rewriter (RR)	The Recommendation Rewriter uses the accumulated results from the previous components. It converts them into TUG and LORE messages that are afterwards send to and presented by the LMS.
Resource Description Framework (RDF)	An URI based data model that allows for the explicit expression of relationships between Web resources in the form of subject – predicate – object triples
Semantic Learning Object Model (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.
Tutorial Guidance (TUG)	The Tutorial Guidance (TUG) is a module in INTUITEL, which enables direct exchange of text, audio and video information between INTUITEL and the learner via the LMS UI, in order to give direct feedback or to request additional information.
User Score Extraction (USE)	The User Score Extraction (USE) is a module in INTUITEL, which supplies the Back End with information about learner performance and their current environmental situation.

2 Introduction

2.1 About this document/Structure

The main goal of this deliverable is to give an overall introduction to the INTUITEL ENGINE. The basic task of the reasoning engine is to create learner specific recommendations and forward them to the Learning Management System. Furthermore, a manual report is provided that offers guidelines to install and deploy the INTUITEL system, showing how the integrated components work together as a unified prototype [DOW p.22]. Therefore, the document is structured as follows:

Chapter 2 gives the motivation and challenges for an ontology-based framework for adaptive Technology Enhanced Learning. Chapter 3 describes the INTUITEL Engine on a conceptual level, i.e. the Query Builder, the Reasoning Broker comprising different reasoners, and the Recommendation Rewriter that consists of a Ranking Module and a Natural Language Unit. Moreover, a new approach to modelling learning pathways is offered, which extends the former Pedagogical Ontology. Chapter 4 is devoted to explain the testing procedures used to evaluate and validate the functionality of the INTUITEL Engine. Chapter 5 offers guidelines to install and deploy the INTUITEL system from a developer's perspective. Then, Chapter 6 discusses the integration of different reasoning engines in a reasoning broker framework, and finally, Chapter 7 highlights the major conclusions.

2.2 Motivation and Vision

In INTUITEL we focus on a personalized and adaptable teaching learning process, focusing of the role of the learner. Accordingly, the advanced 'intelligent' features are that it is

- *user-adaptive,* i.e. it configures itself to any learner. Individual aspects of the learner are considered
- *didactically-enhanced*, i.e. the reasoning takes place on ontologies which represent pedagogical and methodological knowledge

2.3 Problem Statement

The personalized recommendation of learning objects, apart from easy reuse and sharing of such resources, is the primary objective we want to achieve and the focus of this deliverable.

Our basic aim is to provide a high degree of adaptivity and user personalization to gain an optimal learning experience.

We claim that this goal can be reached best by means of a reasoning framework based on formal ontologies that represent the *learner* (user profile and performance) and the *pedagogical strategies*. Since we want to be domain-independent, the integration of concepts and relations for a domain-specific application will be left unspecified as much as possible and needs to be complemented by

end-users. The basic motivation for using formal ontologies in our context is thus the fact that automated reasoning is enabled, based on an explicit specification of knowledge (i.e. according to the semantically enhanced metadata standard SLOM in INTUITEL).

Ontologies provide a uniform model for the entire learning process which can be conceptualized as the navigation through a network of learning objects. At each single step of the process, the learner state will be updated and the learner will be recommended a list of learning objects.

Furthermore, meta knowledge about the reasoning process, i.e. specifically *why* a specific learning object is recommended, will be given as an explanation to the learner in form of a feedback message.

The reasoning-based recommendation process takes place iteratively throughout the learning process. The guiding principle for any recommendation is to check which Learning Objects best match the learner along a set contextual and user profile features (DF-features) which can be static (e.g., gender) or may vary over time (e.g., learning speed). Personalization, in our framework, can thus be understood as checking for DF satisfiability, i.e. given an instance of a user profile of a specific DF type, e.g., DFAge=Senior, the task of the Reasoner is to find an instantiation with concrete Learning Object (LO) values. The choice of the 'user-specific' best pathway will be dependent to a large part on the concrete learners' state, and can only be evaluated dynamically at runtime. Thus, re-computing the best-suited LOs is required every time the user changes his/her learning state.

In this deliverable, we propose a hybrid recommender model using Ontology-based reasoning combined with Simple Additive Weighting for relevance ranking.

2.4 State-of the art: Ontology-based Framework for eLearning

Various ontological frameworks were proposed for e-Learning [15, 14, 28, 9, 24, 5, 17], focusing on incorporation of open standards and personalised recommendations [7, 23].

Knowledge integration in most cases considers the metadata of the learning resources, complemented by a domain model and a learner model [15, 14]. In our work, we explicitly refrain from integrating domain ontologies for the course content since these resources are often not readily available, and building up an ontology from scratch generally requires enormous human effort. We therefore claim that as few assumptions as necessary should be made about the e-Learning domain. On the other hand, we stress the importance of domain-agnostic pedagogical strategies set up by didactic experts. We build up a knowledge base that serves as a playground for tutors to evaluate if the didactic recommendations meet their desires and guide the learner correctly.

Different logical frameworks for ontologies have been used for e-Learning, with varying expressiveness and complexity, ranging from simple taxonomies [13], to more complex representations with axioms that constrain the interpretation of the model. The most common knowledge representation formalisms adopted for e-Learning are F-logic [15, 14, 28, 9], OWL-DL [12], the Semantic Web Rule Language (SWRL) [24], and OWL-S [5]. In our approach, we use OWL 2 DL, a W3C standard that builds on Description Logics and extends the earlier OWL standard by several language features while still preserving decidability.

A fundamental aspect of e-Learning are so-called *Learning Pathways* that aim to optimise the individual learning experience. The most common formal approach to learning paths is based on the IEEE SCORM and IMS simple sequencing specification. However, various different implementations are used.

In ontology-based frameworks, often an additional domain ontology layer is integrated [27, 11] where a *hasResource(C;LO)* relation is used to link a Learning Object to a domain concept. In order to reach a learning goal, the order of concepts is constrained by a partial ordering relation *isRequiredBy(C1;C2).* A similar modelling approach is offered by Yu et al. [30] based on a binary relation *hasPrerequisite(C1;C2)*, which describes content dependency information at the course level to generate a learning pathway.

We offer an extension of the ontology-based approach that takes into account the fundamental characteristics of a learning pathway, such as modular composition, nested composition, optional parts, and sequencing. Thus, our approach is more expressive, since we seek to model entire structured sequences, following the learning pathway specification as suggested by Janssen et al. [16]. Finite state (FS) frameworks also have been used for learning pathway modelling and almost exclusively rely on Directed Acyclic Graphs (DAGs) [2, 17]. The main focus of this work, however, has been on effciency rather than expressiveness with the aim of shortest path analysis, using Weighted FS networks.

The main approach to ranking considers a hybrid approach (i. e., semantic and arithmetic). Shen et al. [27, 11] use competency gap analysis to this aim, favouring learning content that might help the user to progress towards his/her learning goal. Yu et al. [30] give preference to Learning Objects that are in a close taxonomic relationship of the respective domain to the learning goal of the user, specified as *dc:subject* metadata entry.

Both approaches thus require a separate domain model. Opposed to this, we rank the Learning Objects according to the degree of relevance to the learner and select the highest scoring Learning Objects, considering that different Didactic Factors have a different impact on the overall recommendation and to what degree learning objects fulfill the recommendation constraints.

2.4.1 Strength of Ontology-based Frameworks

Information sharing, integration and reuse – Ontologies support the use of well-established standards for defining and sharing Learning Objects within different e-Learning platforms. International metadata standards exist that offer a set of metadata descriptors such as LOM (Learning Object Metadata) and SCORM (Shareable Content Object Reference Model), cf. Aroyo et al. [4], Dolog et al. [9]. Specifically, in complex and different educational environments, this increases interoperability among systems which have to interact and enables reusability of learning material.

Semantic Search and Reasoning – Ontology-based approaches have become increasingly popular, since they offer additional reasoning capabilities and thus support semantic search of LOs [8]. Unlike the search paradigm on the Web, the focus is on searching for structured data, where LOs are semantically annotated on the metadata level. Consequently, more precise information needs can be expressed by means of a complex constraint query. Moreover, LOs are generally related to each

other via structural relationships, i. e., the learning pathways, and this semantic graph can also be exploited for search. However, while in semantic search, users explicitly have to articulate their information need by means of a query, being redefined by more precise queries during the search process, in e-Learning, the educational goal is fixed from the start and the queries are completely hidden from the user.

2.4.2 Weaknesses of Ontology-based Frameworks

Efficiency – Reasoning in expressive Description Logics has exponential runtime complexity in the worst case. However, implementations of state-of-the-art OWL reasoners are typically optimized to show acceptable runtime behaviour in many real-world scenarios. Modelling in the tractable OWL 2 Profiles, however, comes with considerable compromises regarding expressiveness

Requirement for support complex conjunctive queries – Especially in cases where no resource completely satisfies all conjuncts of a given complex conjunctive class expression, it would be of interest to determine the winner among the competing candidates, i. e., the learning resources which fulfil most of the constraints. A naive approach to instance retrieval inference may often return an empty result set, since some constraints might not be satisfied. We thus need to find an optimal solution that satisfies a maximal subset of the constraints.

Sequences – In e-Learning, Learning Objects are organised into sequences that describe an optimal navigational path towards a learning goal. At present, there is no support for defining sequences in OWL as would be needed for reasoning over learning pathways. However, by use of Ontology Design Patterns (ODP), best practice solutions can be adopted that allow for regexp-like queries [10].

2.4.3 Limits of Ontology based Frameworks

Ranked Retrieval – In order to retrieve a ranked list of suitable Learning Objects with a recommendation factor, standard Boolean Retrieval, as facilitated in OWL, is not enough, since it does not support the scoring of objects but delivers an unordered result set. Instead, the results should be ordered and the ranking should reflect the degree of relevance to the learner (according to some given ranking scheme). A semantic search algorithm that integrates ranking based on RDF graphs and similar to the PageRank scoring algorithm has been proposed by Kasneci et al. [19], while an account based on exploiting semantic relationships between entities has also been proposed [1, 3]. While SPARQL allows to rank results by means of the “ORDER BY" predicate [23], the data used for computing the order has to be available in the RDF graph explicitly. To our knowledge, there is no solution to deduce rankings logically in OWL instance retrieval tasks based on multiple criteria.

Support of Soft Constraints – Preferences behave like soft selection constraints. In this sense, no exact match is required and therefore soft constraints should be satisfied if possible, but may be violated if necessary. A metric generally aims to provide an idea for how close a result is to the learners' needs, i. e., to which degree a Learning Object is relevant. The challenge related to the ways in which such vague knowledge in terms of OWL and its reasoners should be incorporated is well recognized, and some fuzzy and rough extensions have been proposed (cf. [20, 26, 22]).

However, a standard semantic web compliant solution regarding vagueness is not available yet. But more importantly, this solution would not be applicable in our approach, since one of the major aims is to verify the pedagogical rules set up by the pedagogical experts, and a fuzzy approach would offer little control over the search and reasoning process.

2.5 Progress beyond the State-of the art

Reasoning & Inference – The INTUITEL Engine computes recommendation candidates based on a novel and enhanced OWL modelling of learning pathways (cf. Section 3.1, 3.3). The Reasoning Broker optimizes the reasoning efficiency by intelligent query building looking at the whole learning process, distributing workload to several reasoners in a broker framework, and using the latest standards for OWL access protocols and caching of result sets (cf. Section 3.3.4, 4.2).

Reasoning & Ranking – An extension to the pure knowledge-based approach is given to overcome the problems of ranking and soft constraints. The calculation for ranking is based on individual weights for the Didactical Factors and a score indicating the degree of similarity/match between learning objects and learner profile. The new methodology is introduced in Section 3.4.2.

Dialogue & Feedback – The INTUITEL communication layer forwards the feedback messages provided by the back end to the user, focusing on meta-cognitive feedback in terms of motivational messages and giving explanations why a specific recommendation was given. Furthermore, dialogue messages are triggered in case of missing (meta-)information (e.g. user did not specify his/her age). On top of the standard template-filling approach, we provide an enhanced Natural Language Generation (NLG) component to make the tutorial dialogue appear more natural (cf. Section 3.4.3).

3 Conceptual and technical definition of the INTUITEL Framework

This chapter introduces the INTUITEL framework with a special focus on the WP05 elements. The INTUITEL framework (schematically depicted in Figure 1 and described in more detail in the INTUITEL System Design Document specification D3.1) consists of a loosely coupled set of components, which interact in a defined order. The individual components are event-based and get active in reaction to a user action, e.g. when a learner works on a course in the Learning Management System (LMS).

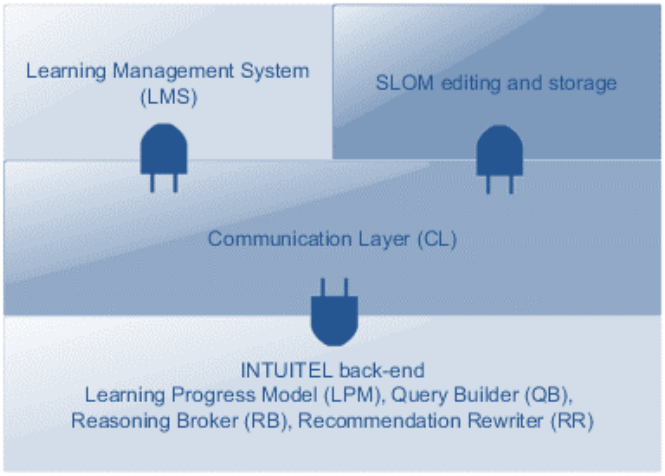


Figure 1: High-level framework overview

The data exchange is managed by the INTUITEL Communication Layer (CL). The message format is XML and all messages satisfy a common XML schema definition[3]. For this purpose, a Java API, which encapsulates all messages, has been created to accompany the CL. This approach ensures that the components run independently of each other and can focus on their core tasks.

The INTUITEL back-end is a constructed of three parts, which together build a data pipeline for the processing of learner data:

1) The Learning Progress Model (LPM)
2) The Query Builder (QB) and Reasoning Broker (RB)
3) The Recommendation Rewriter (RR), containing the Natural Language Unit (NLU) and the Ranking Module (RM)

In combination with SLOM, those modules create learning recommendations and feedback messages. The Query Builder, Reasoning Broker and Recommendation Rewriter comprise the

[3] Learning Management System: http://www.intuitel.eu/public/intui_DMLMS.xsd
Learning Progress Model: http://www.intuitel.eu/public/intui_DMLPM.xsd
Query Builder: http://www.intuitel.eu/public/intui_DMQB.xsd
Recommendation Rewriter: http://www.intuitel.eu/public/intui_DMRR.xsd
SLOM repository: http://www.intuitel.eu/public/intui_DMSLOMREPO.xsd

INTUITEL Engine (i.e. WP05). Although the LPM is not part of WP05, it will also be covered here briefly to give a comprehensive guide to the INTUITEL back-end.

Stage 1 - Learning Progress Model: The LPM is a Maven project and is the main entry point for data originating from the LMS. It takes care of data storage and pre-processes the accumulated information in form of a learner-specific ontology, the Learner State Ontology (LSO)[4]. The creation of LSOs is parallelized in "LPM processors" which each run in an own thread and, amongst others, determine the currently valid Didactic Factors (DFs) of the learner.

Stage 2 - Query Builder & Reasoning Broker: The QB and the RR are both Maven projects that work hand-in-hand. The QB receives the LSOs as created by the LPM and enriches them with additional predefined axioms that allow a reasoner to determine the suitable learning objects for a learner. To do this, it also needs the SLOM data of the respective course, which it retrieves from the SLOM repository. The Reasoning Broker takes care of the selection of suitable reasoners and parallelization of reasoning tasks to optimize the system's performance.

Stage 3 - Recommendation Rewriter: The RR also is a Java program (ANT-based) which uses the accumulated results from the previous components. It converts them into TUG and LORE messages that are afterwards send to and presented by the LMS. Two sub-components are active in this process. The NLU creates the text messages for TUG, based on Reflexes and Didactic Factors, and the RM uses the Didactic Factor data to prioritize the reasoning results for LORE messages.

In all stages, a special emphasis was put on adjustability and extendibility of the system. It is thus possible for users of the INTUITEL system to influence what the system takes into account and the respective degree of importance. This means that (i) the weights of individual DFs can be adjusted, or (ii) new DFs can be added without impairing the overall functionality of the system.

3.1 Preliminaries

3.1.1 Summary of INTUITEL Ontologies

We propose a modular ontology design in order to cover static pedagogical background knowledge as well as course and learner specific, dynamic knowledge.

INTUITEL OWL Ontologies:

- The **Pedagogical Ontology (PO)** defines the pedagogical background knowledge and has concepts w.r.t. difficulty of course material, relevance, knowledge type, media type and sequencing rules. Learning material is organized into Knowledge Domains (KDs), Concept Containers (CCs), and Knowledge Objects (KOs), forming a hierarchical graph structure (see D2.1).

[4] For more details on the LSO, please see Deliverable 3.2 (http://www.intuitel.de/fileadmin/resources/INTUITEL_318496_D3_2_LPMSpecDocument.pdf)

- The Learner Model Ontology **(LMO)** defines classes and properties for describing the current learner state (see D3.2).
- The **Cognitive Map (CM)** characterizes a particular domain of knowledge (see D2.2), including macro LPs.
- The course-specific **Cognitive Content Model** (CCM) is an instantiation of the Pedagogical Ontology which comprises the Knowledge Objects and their metadata as well as the micro LPs.
- The **Learner State Ontology (LSO)** contains learner-specific data as instances of the Learner Model Ontology. A snapshot is characterized by Didactic Factors (DFs) that currently hold, including the completion state of KOs and CCs, and current learning pathways. Since the user model is updated automatically by tracking users' behaviour (assessment results, clickstream, LOs visited, etc.), the LSO is time and user-specific.

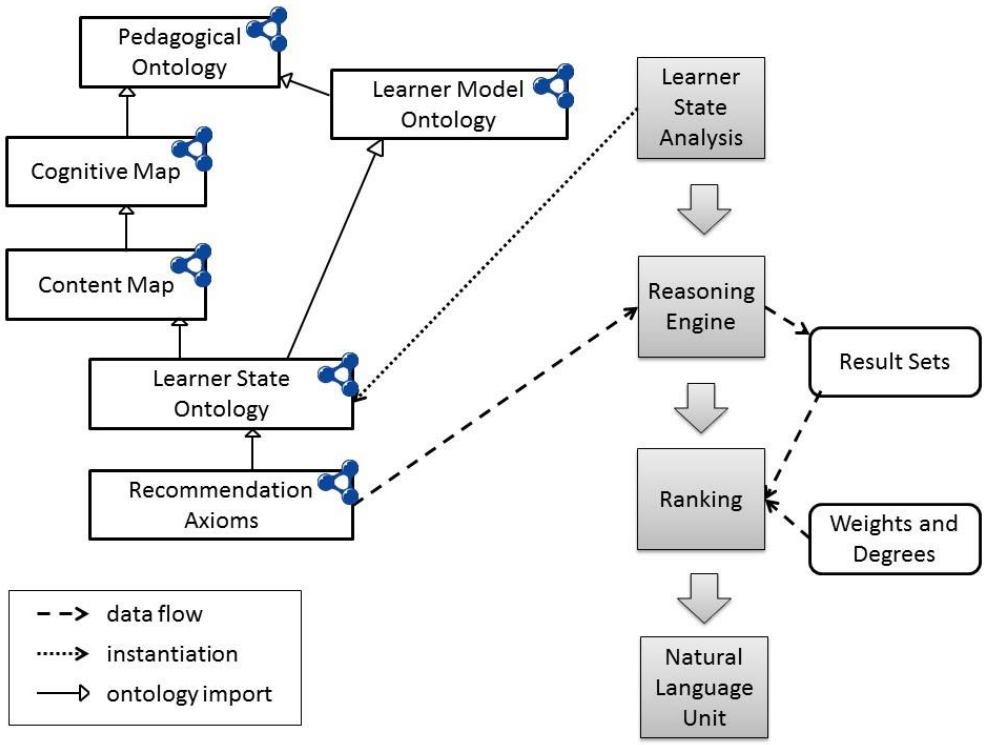


Figure 2 INTUITEL Ontology-based Reasoning Framework

3.1.2 Novel Approach to Modelling Learning Pathways

Learning pathways are specified as macro-level learning pathways on the level of CCs, and micro-level learning pathways on the level of KOs in D2.1 in the Pedagogical Ontology [29]. Technically, i.e. in terms of an ontology model in OWL, these learning pathways were represented as object properties connecting any two KOs (CCs, resp.), which are in a direct successor relationship with regard to a particular learning pathway. Since OWL does not support the representation of n-ary relations, it is not possible to specify generic learning pathway object properties that can be reused

for different pathways. Moreover, it is not possible to attach any additional context to a specific pathway relation between any KOs (CCs, resp.), for instance, to specify that a particular micro-level learning pathway relation between two KOs only hold in a particular CC. The consequence was that every learning pathway required its own object property for denoting successor relations, which was defined in the Pedagogical Ontology. In order to increase flexibility in terms of allowing tutors to individually specify learning pathways without having to alter the (generic) Pedagogical Ontology or modifying the ontology vocabulary (schema) by creating new object properties, the INTUITEL consortium agreed on an alternative approach for modelling learning pathways.

In the new approach, learning pathways are no longer represented as object properties, but as ontology classes. There are generic ontological entities defined in the Pedagogical Ontology, which are to be extended / instantiated in the CM or CCM, where specific learning pathways are defined. These generic entities are:

- *MacroLearningPathway* – Upper level class to describe macro-level learning pathways. Cognitive engineers are expected to create subclasses of this class for each specific macro-level learning pathway they create for their Cognitive Map. Every connection between any two CCs with regards to this specific macro-level learning pathway is represented by a connector individual as instance of this class.
- *MicroLearningPathway* – Upper level class to describe micro-level learning pathways. Cognitive engineers are expected to create subclasses of this class for each specific micro-level learning pathway they create for their Cognitive Content Map. Every connection between any two KOs with regards to this specific micro-level learning pathway is represented by a connector individual as instance of this class.
- *hasPredecessorInMacroLp* – Generic object property connecting an individual of a MacroLearningPathway-subclass to the CC which acts as the predecessor in a macro-level learning pathway relation.
- *hasSuccessorInMacroLp* – Generic object property connecting an individual of a MacroLearningPathway-subclass to the CC which acts as the successor in a macro-level learning pathway relation.
- *hasPredecessorInMicroLp* – Generic object property connecting an individual of a MicroLearningPathway-subclass to the KO which acts as the predecessor in a micro-level learning pathway relation.
- *hasSuccessorInMicroLp*. – Generic object property connecting an individual of a MicroLearningPathway-subclass to the KO which acts as the successor in a micro-level learning pathway relation.

This modelling pattern allows for a flexible representation of learning pathways, where each learning pathway is represented as a set of individuals, each denoting a successor-relation between two KOs (CCs, resp.).

Formally, to connect two KOs, KO_i and KO_j via a micro-level learning pathway, *MyMicroLP*, using a connector individual $CKO_{(i,j)}$, the following ontology axioms are required:

$$MicroLearningPathway \sqsubseteq LearningPathway$$
$$MyMicroLP \sqsubseteq MicroLearningPathway$$
$$MyMicroLP(CKO_{(i,j)})$$
$$hasPredecessorInMicroLp(CKO_{(i,j)}, KO_i)$$
$$hasSuccessorInMicroLp(CKO_{(i,j)}, KO_j)$$

With different learning pathways available, it must be made known to the INTUITEL Engine, which is the one currently chosen by the learner. This information is part of the learner state and can be represented by making the according learning pathway a subclass of the class *CurrentLearningPathway*.

$$CurrentLearningPathway \sqsubseteq LearningPathway$$
$$MyMicroLP \sqsubseteq CurrentLearningPathway$$

Due to the open world assumption of OWL, it is necessary, to mark the first and last pathway element, in order to know where to begin a Lesson (CC), and when to move on to the next Lesson (CC).

$$FirstLPElement(CKO_{(i,j)})$$
$$LastLPElement(CKO_{(k,l)})$$

3.1.2.1 Explicit Specification of Learning Pathways

Following this modelling pattern, a learning pathway can be explicitly specified by the following steps:

1. Creating a subclass of *MicroLearningPathway* (*MacroLearningPathway*, resp.) in the Cognitive Content Map (Cognitive Map, resp.)
2. For each connection between two KOs (CCs, resp.) according to the new learning pathway, create an individual as an instance of the new subclass
3. Create assertions for object properties pointing from the newly created individual to the predecessor and successor KOs.
4. Assert the first and last connector individual to the classes *FirstLPElement* and *LastLPElement*, resp.

This process can be eased by appropriate authoring tool support, such as the INTUITEL Editor, or transformation scripts.

3.1.2.2 Inferring Learning Pathways based on Knowledge Type / Media Type pathways

Apart from explicitly specifying learning pathway on the learning object level, it is possible to infer pathways based on didactic theories on knowledge type pathways or media type pathways [29]. Based on the OWL semantics, these inferences can be accomplished by specifying the knowledge type (media type, resp.) for each KO, provided that generic didactical knowledge type (media type) pathways are modelled as background knowledge in the Pedagogical Ontology.

The realization required only few additional entities in the Pedagogical Ontology (only the modelling of Knowledge Type pathways is discussed here, the case for Media Type pathways is analogous):

- *KnowledgeTypeLearningPathway* - Upper level class to describe knowledge type learning pathways. Didactical experts create subclasses of this class for each specific knowledge type learning pathway according to didactic research. Every connection between any two Knowledge Types with regards to this specific knowledge type learning pathway is represented by a connector individual as instance of this class. In contrast to explicitly modelled learning pathways, this representation is part of the generic Pedagogical Ontology.
- *hasPredecessorInKTLp* - Generic object property connecting an individual of a KnowledgeTypeLearningPathway-subclass to the Knowledge Type which acts as the predecessor in a knowledge type learning pathway relation.
- *hasSuccessorInKTLp* - Generic object property connecting an individual of a KnowledgeTypeLearningPathway-subclass to the Knowledge Type which acts as the successor in a knowledge type learning pathway relation.
- *hasKnowledgeType* – Generic object property connecting a KO to a Knowledge Type.

The actual pathways on the knowledge type level are specified in the same way as macro- or micro-level pathways, but generically by didactic experts. Formally, to connect two Knowledge Types KT_i and KT_j via a knowledge type learning pathway, *SimulatedMultiStage*, using a connector individual $CKT_{(i,j)}$, the following ontology axioms are required:

$$KnowledgeTypePathway \sqsubseteq MicroLearningPathway$$
$$SimulatedMultiStage \sqsubseteq KnowledgeTypePathway$$
$$SimulatedMultiStage(CKT_{(i,j)})$$
$$hasPredecessorInKTLp(CKT_{(i,j}, KT_i)$$
$$hasSuccessorInKTLp(CKT_{(i,j}, KT_j)$$

Provided, the Knowledge Type is specified for each KO via the object property *hasKnowledgeType* the predecessor and successor relations on the KO level can be inferred via the following property chain:

$$hasSuccessorInKTLp \circ hasKnowledgeType^{-} \sqsubseteq hasSuccessorInMicroLp$$
$$hasPredecessorInKTLp \circ hasKnowledgeType^{-} \sqsubseteq hasPredecessorInMicroLp$$

Setting the currently chosen learning pathway to a knowledge type pathway, allows for inferring micro-level learning pathways based on the didactical theories encoded as knowledge type pathways in the Pedagogical Ontology without any further modelling efforts.

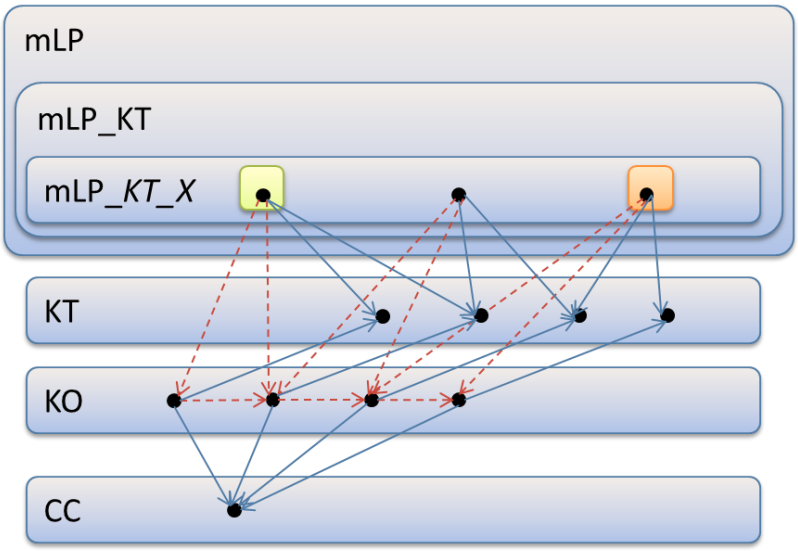


Figure 3: Modelling pattern for knowledge type learning pathway successor relations.

The modelling pattern for knowledge type pathways is illustrated in Figure 3. Subclasses of mLP_KT (Knowledge Type learning pathway) denote specific pathways. Instances of these classes are used to connect any two Knowledge Types via hasPredecessor and hasSuccessor relations. First and last pathway elements are marked by additional classes (green boxes for first pathway elements, and red boxes for last pathway elements.) hasPredecessor and hasSuccessor relations on the KO level are inferred from the Knowledge Type pathways and the hasKnowledgeType assertions for the KOs.

3.2 INTUITEL Architectural Design

The INTUITEL Communication Layer not only acts as a mediator between the three core parts of the overall system (i.e. an enhanced LMS, SLOM editing and storage) as well as the INTUITEL back-end, but also between the individual back-end components (cf. Figure 14 “Architectural overview” and Figure 15 “Sequence Diagram” in the Appendix).

The advantages of this approach are (i) the event-based characteristic of the system is preserved internally too, (ii) each of the modules can be developed as a stand-alone application, (iii) the modules remain independent of each other and can easily be extended, replaced or updated without affecting the others, (iv) the components can be run on different machines to make the solution better scalable and (v) existing solutions can be re-used.

The only exception to this paradigm is the back-end database, the UserDb, which contains the learning data as logged by the LPM. While this information in mainly relevant for calculating the current learner state and the LSO, the RR also requires access to it in order to create TUG messages and to store the replies of the learner (e.g. in order to store a LP selection, there needs to be some kind of access to the data storage).

As noted in the introduction to this chapter, the back-end works in three stages. This holds true for the usual scenario of creating learning recommendations. However, there are cases where the system deviates from this routine (e.g. when important information for a recommendation is missing). In those situations, there simply is not enough data to properly reason on and triggering the respective component would thus be unnecessary. The usual process is then cut short using a reflex to directly send information to the RR, which then creates a proper response in form of a TUG message. The most common reflex is missing LP selections. Before a learner gets a recommendation, the INTUITEL system needs to know which micro and macro LP the learner wants to use. To give learners a selection possibility, the LPM creates a reflex and the RR thereafter generates a respective TUG message. When the learner chose and a response was received, the RR stores it in the UserDb and triggers with the LMS a refresh of the last learner update at the LPM, consequently leading to a learning recommendation.

3.2.1 Communication Layer

The INTUITEL Communication Layer (CL) is the cornerstone for the messages exchange between all components of the INTUITEL System, being these components either internal such as the Engine or the SLOM repo or external such as the different LMSs. During the INTUITEL project, two versions of the CL have been developed: a Basic CL and an Advanced CL. However, the way they can be accessed is equivalent, although the installation and configuration are completely different each other .

The Communication Layer (CL) is deployed following two different versions:

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

The type of CL does not affect the functionality and the configuration of the other INTUITEL components; therefore, it is possible to switch between the two CL versions without any problem. A comprehensive description of the CL is given in D 3.3.

3.2.2 User Database

The INTUITEL User Database is used by two different components: the LPM and the Recommender Rewriter. The main role of this component is the storage of all the dynamic parameters, such as user profile, session data and recommendations.

3.2.2.1 Quick requirement analysis

The INTUITEL Database needs to store all the relevant data processed by the LPM, therefore several data structures are needed:

a) **Learner profile** and **preferences**. This structure needs to model the user with respect to his privacy. It needs to contain data about the gender, age, spoken languages, learner's culture and accessibility settings. Moreover, the user profile should contain the preferred European Qualifications Framework (EQF) and difficulty levels for each course. Some of this data is updated via the USE interface of the LMS, whereas the difficulty levels are self-assessed by the learner and updated via the TUG interface.

b) A learning **Session** contains static elements (device type, resolution, interaction willingness and learning attitude) which do not change over a session, and, dynamic data (connection type, noise and battery level, connection stability) which changes over time and is updated over a sampling interval. This data is provided by the LMS via the USE interface.

c) **Learning Pathways** and **Recommendations**. These data structures are used by the LPM to store all the relevant recommendation elements, such as the current micro and macro learning pathway selection, the scores and completion rate for each concept container and the knowledge object accessed by a learner.

d) **LMS Profile.** Currently, the INTUITEL Back End accepts only one LMS connection. Nevertheless, the LMS parameters could change over time, which could lead to some configuration issues. To prevent these problems, the database should store the current LMS profile for future reference.

3.2.2.2 Current design

The current design chosen for the INTUITEL database uses a standard SQL structure, which can be deployed on almost all the common SQL databases and dialects. Figure 16 (cf. Appendix) shows the current design, based on a quick requirement analysis.

From the technical perspective, a MySQL Community[5] (version 5.6.20 for Windows) was chosen. There are several deployment options available and the current database design has been successfully tested with EasyPHP[6], XAMPP[7] bundle and MySQL Workbench[8]. This assures the independence from the database toolkit and assures a multi-platform standard.

[5] MySQL Community Edition: http://www.mysql.com/products/community/
[6] EasyPHP: http://www.easyphp.org/

The database component comes bundled with a Java library to enable easy integration with the whole INTUITEL architecture. This has been developed with the Hibernate API[9], a Java library that can be used to develop robust and database-independent serialisation layers. Choosing this technology avoids the need for using raw SQL scripts in the Java code, which decreases the risk of having SQL syntax errors in deployed environments.

The Java library can be used either as a Maven project (current LPM integration) or an independent Ant built library (used with the Recommender Rewriter). Moreover a full deployment and integration guide is provided in the setup section (Section 5.3).

3.3 INTUITEL Query Builder & Reasoner

3.3.1 Overview

The INTUITEL Query Builder is a component within the INTUITEL Engine that computes sets of KOs as recommendation candidates. Each set represents a selection of KOs according to a particular selection criterion. There are three kinds of selection criteria:

1. Selection of KOs based on the currently chosen learning pathways
2. Selection of KOs based on Didactic Factors which must be fulfilled (hard criteria)
3. Selection of KOs based on Didactic Factors which should be fulfilled (soft criteria)

Each selection criterion is encoded as an OWL class expression, describing the characteristics that need to hold for any individual in order to be a member of it. The Query Builder combines the Learner State Ontology (which itself contains the Pedagogical Ontology, as well as instantiations regarding a specific Cognitive Map and Cognitive Content Map) with generic query axioms and class expressions, and invokes an OWL reasoning framework in order to retrieve instances for the relevant class expressions. The generic query axioms and class expressions are externalized as an ontology being part of the Query Builder, and which the Query Builder merges with the Learner State Ontology it receives as input from the LPM. This makes the INTUITEL Engine a powerful and flexible knowledge-based system, where complex descriptions of learning pathways and Didactic Factors can be used to compute recommendation candidates.

3.3.2 KO selection based on learning pathways

Due to the formal, Description Logics based semantics of OWL, sets of recommendation candidates can be computed by an OWL reasoning framework using logical deduction.

---

[7] XAMPP: https://www.apachefriends.org/
[8] MySQL Workbench: http://www.mysql.com/products/workbench/
[9] Hibernate Java API: http://hibernate.org/

3.3.2.1 Axioms and class expressions

Recommendation candidates based on learning pathway relations require knowledge about the KO currently being visited by the learner, as well as the KO that was visited by the learner before the current one. This information is part of the learner state and formalized as class assertions in the LSO:

$$CurrentKO(KO_i)$$
$$PreviousKO(KO_j)$$

This knowledge allows for the computation of sets of KOs based on their position on the learning pathway relative to the current, resp. previous KO.

Relevant for being a member of a set of candidate recommendations is the completeness state of a KO. To this end, the states "completed", "partially completed", and "unseen" are distinguished. The states are represented as OWL classes that form a complete segmentation of the class *LearningObject*, i.e. the class *LearningObject* is defined as the disjoint union of those three classes, formalized as follows:

$$LearningObject \equiv CompleteLO \sqcup PartiallyCompleteLO \sqcup UnseenLO$$
$$CompleteLO \sqcap PartiallyCompleteLO \sqsubseteq \bot$$
$$CompleteLO \sqcap UnseenLO \sqsubseteq \bot$$
$$PartiallyCompleteLO \sqcap UnseenLO \sqsubseteq \bot$$

Logical inference of pathway successors using the learning pathway modelling pattern can be accomplished via OWL property chain axioms. Since the recommendations are expected to refer to the currently chosen learning pathway, the knowledge about a learning pathway being the current one has to be available via an object property, such that it can be used in a property chain. This can be achieved by the following axiom containing a Self-restriction:

$$CurrentLearningPathway \sqsubseteq isCurrentLP.\boldsymbol{Self}$$

This logically entails that every connector individual which is a member of the class *CurrentLearningPathway* is linked via an object property *isCurrentLP* to itself.

The following property chain allows for inferring the direct successors of any KO with regards to the current micro-level learning pathway:

$$hasPredecessorInMicroLp^{-} \circ isCurrentLP \circ hasSuccessorInMicroLp \sqsubseteq hasDirectKOSuccessor$$

This solution is already sufficient in order to infer micro-level learning pathway successors within one CC, but not across CCs, and thereby following the macro-level learning pathway. In order to consider the special case of micro-level learning pathway successor at the end of a CC, the following additional axioms are required.

Firstly, a property chain is required for inferring direct successors of CCs with regards to the current macro-level learning pathway, analogous to the property chain for micro-level learning pathways:

$$hasPredecessorInMacroLp^{-} \circ isCurrentLP \circ hasSuccessorInMacroLp \sqsubseteq hasDirectCCSuccessor$$

(Note that the modelling pattern for specifying the current learning pathway covers micro-level and macro-level learning pathways likewise.)

Secondly, an auxiliary object property is required, that connects each KO in a CC to all KOs in the CC that is the direct successor with regards to the current macro-level learning pathway:

$$isContainedByCC \circ hasDirectCCSuccessor \circ isContainedByCC^{-} \sqsubseteq nexKOsInNextCCs$$

In case not only direct pathway successors, but all successors (direct and indirect) are required, the concept of transitive superproperties can be used. For instance, an object property relating a KO to all its successors with regards to the current learning pathway, *hasKOSuccessor*, can be specified as follows:

$$hasDirectKOSuccessor \sqsubseteq hasKOSuccessor$$
$$\boldsymbol{trans}(hasKOSuccessor)$$

Another auxiliary construct is a class expression describing all KOs on the current micro-level learning pathway:

$$AllKOsOnCurrentMicroLP \equiv$$
$$\exists hasPredecessorInMicroLp^{-}.CurrentLearningPathway \sqcup \exists hasSuccessorInMicroLp^{-}.CurrentLearningPathway$$

3.3.2.2 Recommendation candidate sets

There are four learning pathway related sets of recommendation candidates computed based on the successor / predecessor relations and current / previous KOs.

1. **Direct successors of the current KO (Recommendation Axiom 1.1)**

   This is the set of KOs which are direct successors of the current KO according to the current micro-level learning pathway. If the current KO is indicated as the last KO on the current micro-level learning pathway in the current CC, the set will contain the KOs which are indicated as first ones on the current micro-level learning pathway in the subsequent CC according to the current macro-level learning pathway. Only KOs are being considered that do not have the status “complete” (i.e. that are “partially complete” or “unseen”).

   Class expression:

$$\exists hasDirectKOSuccessor^{-}.CurrentKO \sqcup \Big(\exists nextKOsInNextCCs^{-}.\big(CurrentKO \sqcap \exists hasSuccessorInMicroLp^{-}.(CurrentLP \sqcap LastLPElement)\big) \sqcap \big(\exists hasPredecessorInMicroLp^{-}.(CurrentLP \sqcap FirstLPElement)\big)\Big) \sqcap \neg CompleteLO$$

2. **All successors of the current KO (Recommendation Axiom 1.2)**

   This is the set of KOs which are direct or indirect successors of the current KO according to the current micro-level learning pathway in the current CC. If the current KO is indicated as last KO on the current micro-level learning pathway in the current CC, the set will contain all KOs on the current micro-level learning pathway in the subsequent CC according to the current macro-level learning pathway. Only KOs are being considered that do not have the status “complete” (i.e. that are “partially complete” or “unseen”).

   Class expression:

$$AllKOsOnCurrentMicroLP \sqcap \Big(\big(\exists nextKOsInNextCCs^{-}.\big(CurrentKO \sqcap \exists hasSuccessorInMicroLp^{-}.(CurrentLearningPathway \sqcap LastPathwayElement)\big)\big) \sqcup \exists hasKOSuccessor^{-}.CurrentKO\Big) \sqcap \neg CompleteLO$$

3. **All predecessors of the current KO (Recommendation Axiom 1.3)**

   This is the set of KOs which are direct or indirect predecessors of the current KO according to the current micro-level learning pathway in the current CC. If the current KO is indicated as first KO on the current micro-level learning pathway in the current CC, the set will be empty, i.e. it will *not* contain all KOs on the current micro-level learning pathway in the previous CC according to the current macro-level learning pathway. Only KOs are being considered that do not have the status “complete” (i.e. that are “partially complete” or “unseen”).

   Class expression:

$$AllKOsOnCurrentMicroLP \sqcap \exists hasKOSuccessor.CurrentKO \sqcap \neg CompleteLO$$

4. **Direct successors of the previous KO (Recommendation Axiom 1.4)**

   This is the set of KOs which are direct successors of the previous KO according to the current micro-level learning pathway. If the previous KO is indicated as the last KO on the current micro-level learning pathway in the current CC, the set will contain the KOs with are indicated as first ones on the current micro-level learning pathway in the subsequent CC according to the current macro-level learning pathway. Only KOs are being considered that do not have the status “complete” (i.e. that are “partially complete” or “unseen”).

   Class expression:

$$\exists hasDirectKOSuccessor^{-}.PreviousKO \sqcup \Big(\exists nextKOsInNextCCs^{-}.\big(PreviousKO \sqcap \exists hasSuccessorInMicroLp^{-}.(CurrentLP \sqcap LastLPElement)\big) \sqcap \big(\exists hasPredecessorInMicroLp^{-}.(CurrentLP \sqcap FirstLPElement)\big)\Big) \sqcap \neg CompleteLO$$

3.3.3 KO selection based on Didactic Factors

Since OWL class expressions allow for the description of criteria that need to be fulfilled by any member of that class, it allows for specifying the properties any KO needs to have in order to be in a recommendation candidate set. Each Didactic Factor is a definition of such criteria. Didactic Factors are defined as OWL class expressions in the Learner Model Ontology (LMO), which is imported by the Learner State Ontology (LSO) and thus available to the Query Builder. The Query Builder extracts those class expressions from the LSO and executes instance retrieval tasks using the reasoning framework.

The Query Builder evaluates the class expressions, regardless their priority as hard or soft criteria, but passes this information on to the Recommender.

3.3.4 Reasoning Broker Framework

The Query Builder, as a knowledge based system, takes advantage of a flexible and abstract representation of knowledge artefacts and relationships, and avoids the use of static rules and application logic that is hard to maintain. Instead, a generic OWL reasoning Back End is used in order to compute instance sets. The declarative approach of describing sets of KO recommendation candidates using OWL requires the use of a robust and scalable OWL reasoning framework.

The HERAKLES Reasoning Broker Framework initially designed and developed in the THESEUS research program funded by the German Federal Ministry of Economics and Technology (BMWi) is such a framework [31]. The original prototype from 2012 was designed for a static use case environment, and thus was scalable and fault tolerant, but not very flexible regarding changing workload and reasoner recovery.

In INTUITEL, the HERAKLES Reasoning Broker was extended by the following features:

- Intelligent HERAKLES Server component.

  In the first version of the HERAKLES Reasoning Broker, external reasoners had to be managed manually on remote servers. Hence, on each server, one or several OWLlink reasoning servers had to be started, and their host/port address had to be made known to the HERAKLES client, manually. In case a remote reasoner crashed, it was impossible for the broker to restart it, or launch additional reasoners on other servers to replace the lost reasoner.

  The extensions made to the HERAKLES Reasoning Broker in the context of the INTUITEL project address this issue by introducing a new HERAKLES Server component. The HERAKLES server is implemented as a REST server, which controls various OWLlink reasoning servers on a particular physical or virtual machine. This allows the HERAKLES client to remotely start and stop OWLlink reasoners on demand.

- Analyser, Loader, and Executor

  The first version of the HERAKLES Reasoning Broker implemented query delegation via a load strategy and an execution strategy. This concept was redesigned in the context of the

INTUITEL project by introducing the Analyser as additional component, and thus splitting the responsibility of the loading strategy into Analyser and Loader. To this end, the Analyser analyses the expressiveness of the given ontology and query, and matches it with the capabilities of available reasoners. The Loader remotely launches suitable reasoners dynamically via the HERAKLES Server component. The Loader is also capable of launching additional reasoners in case higher reasoning capacity is needed.

Deploying the HERAKLES Reasoning Broker as reasoning Back End in the INTUITEL Engine has several advantages compared to using a static reasoning engine:

- The INTUITEL Engine is self-adaptive regarding the number of learners using the INTUITEL-enabled LMS at the same time. In case a large number of learners are using the system in parallel, the HERAKLES Reasoning Broker is able to dynamically launch additional reasoners in order to ensure acceptable response times.

The INTUITEL Engine is flexible with respect to future extensions in terms of pedagogical knowledge and learner model descriptions. Additions and changes in the pedagogical and learner model, which might occur in the future, can result in a change of requirements for the reasoning engine. This is due to the fact that reasoners are typically optimized to perform well for ontologies with particular expressiveness (languages features that are used). Changing the expressiveness, e.g. by altering the pedagogical and/or learner model, might result in a need to exchange the reasoning engine, in order to ensure optimal performance. The HERAKLES Reasoning Broker makes use of its Analyser component to match the ontology expressiveness to the capabilities and strengths of different types of reasoners, and selects the most suitable ones dynamically. Figure 4 illustrates the architecture of the HERAKLES Reasoning Broker and how it is integrated with the INTUITEL Query Builder component. Several HERAKLES Clients act as workers for processing concurrent learner requests. A pool of HERAKLES Servers (possibly using a distributed IT infrastructure) is available for all workers. Each worker can request their own set of OWLlink reasoning servers via the HERAKLES Servers.

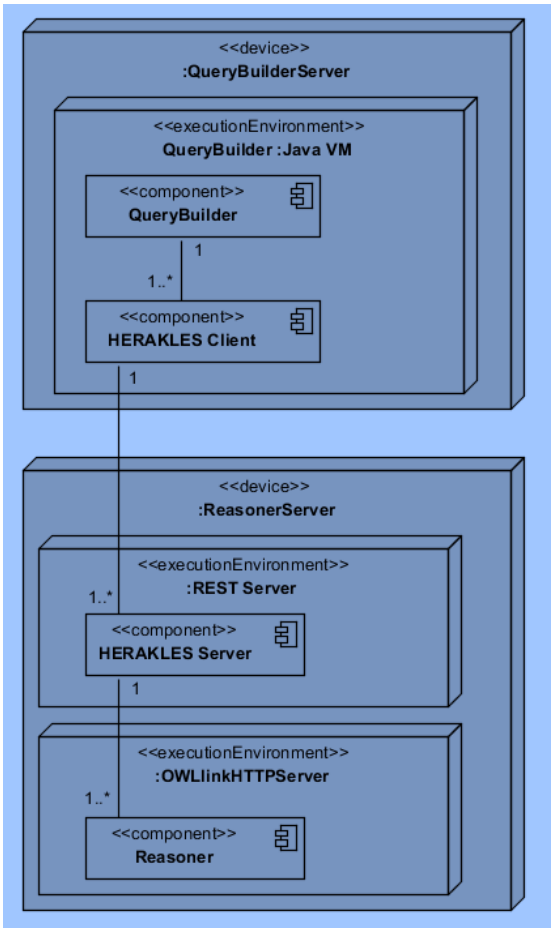


Figure 4: Deployment diagram of the HERAKLES Reasoning Broker integrated with the Query Builder

3.4 INTUITEL Recommender

3.4.1 Overview

The INTUITEL Recommender (RR) component needs to transform all the internal engine data into user-friendly natural language messages and content recommendations. In the current system architecture, the Recommender becomes a service fully integrated with the communication layer. The input data for this service consists of the reasoning results received from the Query Builder, Reflex messages sent by the LPM and certain database elements.

After a brief requirement analysis, it was determined that the Didactic Factors, User Parameters (Profile, Preferences) and Learning Objects are inputs for the Recommender. The Didactic Factors are computed by the LPM and forwarded by the Query Builder, along with the Learning Objects, to the Recommender. The User Parameters are retrieved from the user database, such as age, gender, current preferred difficulty level or current selected course. In addition, the component should provide feedback messages corresponding to any combination of inputs. Moreover, some messages and parameters are more important than the others and might require an answer from the user.

In conclusion, two different sub-components were identified: a feedback and a ranking module. The ranking engine has the non-trivial task of rank-ordering the recommended learning objects based on the Didactic Factors and learning pathways. The feedback module looks like a classic dialogue management system, but it is not. It is a system where the user does not have the initiative and all the requests come from a back-end engine. Nevertheless, the dialogue management task remains non-trivial. Figure 17 in the Appendix shows the general architecture of this component along with its internal dataflow.

3.4.2 Ranking Module

3.4.2.1 Basic Idea

The INTUITEL Ranking Module is a component that builds on the results provided by the INTUITEL Query Builder and the LPM to retrieve the final result, i.e. a personalized recommendation of suitable learning objects. The basic idea behind this module is twofold. When run as in normal mode termed "HardRanking", different result sets will be re-combined (fast intersections are calculated), thus avoiding complex conjunctive queries during reasoning. Moreover, when applied in an advanced mode termed "SoftRanking", ranking of recommendation results can be triggered. To this end, different weights are assigned to Didactic Factors that indicate their relative importance and impact on the overall result and translate into scores that can be exploited for ranking.

3.4.2.2 Formal Approach

In INTUITEL, we adopt the multi-attribute utility theory (MAUT) for ranking as an additive, multi-linear function that maximizes preferences or fitness

- *The utility is determined by the values x = [x1, …, xn] of multiple attributes X = X1, …, Xn*
- *Each attribute is defined in such a way that higher values of the attribute correspond to higher utilities*
- *If for a pair of attribute vectors x and y it holds that xi ≥ yi . For all i, then x strictly dominates y*

3.4.2.3 Definition of the Utility Function

In our work, the highest scoring "most recommended" learning object is calculated based on the MAUT model, combining all DF weights to an overall score.

The basis for ranking is provided by

1. Degree of Match. Parameter d is used to define when constraints given as a key value pairs match. For instance, a Portuguese learning object can be recommended to a native speaker of Spanish because it tends to be comprehensible, even though it does not match the learner profile perfectly.

2. Weights describing the importance of a Didactical Factor. Different weights can be assigned (by the tutor) to individual features, reflecting their importance with respect to all

other feature constraints. For instance, the DF "gender" in most pedagogical frameworks seems to be of minor importance for recommendations.

We use the following Recommendation score to calculate the suitability of a learning object for a certain learner in a certain context:

$$\text{RecommendationScore (KO i)} = \sum_{k=1}^{n} w(k).d(i,k) \qquad (1)$$

where

$w(k)$ = weight of the Didactical Factore k, and thus contribution to the final result

$d(i,k)$ = matching degree of Didactical Factor k, represented by a floating-point value ranging from 0 and 1

n = number of Didactical Factors

Accordingly, the results that best match the axiom (figuring the learners' needs) are ranked higher in the list. The ranking algorithm takes into account the weight values for each Didactical Factor and looks for the highest overall score, so that the number of DFs that are satisfied can still be low, if they are assigned a high overall weight. Since the weight values are subjective, we allow to manually editing them. In our framework, all DF features are independent of each other.

3.4.2.4 Implementation of the Ranking Function

The INTUITEL Ranking is implemented in Java and uses the intermediate result sets form the LPM and Query Builder, encoded as XML elements:

1. INTUITEL LPM: Set of instantiated Didactic Factors for a particular learner
2. INTUITEL Query Builder: Set of successor, predecessor, etc. learning objects based on the currently chosen learning pathway and chosen Axiom (cf. 3.3.2.2)
3. INTUITEL Query Builder: For each Didactic Factor and instantiated value, a set of learning objects that fulfil the class expression (cf. 3.3.2.2)

Moreover, for the advanced *SoftRanking* mode, the following parameters are required:

1. INTUITEL Ranking: For each Didactic factor and instantiated value, a value for the DF weight
2. INTUITEL Ranking: For each Didactic factor and instantiated value, a value for the degree of match depending on the specific learning object metadata.

An example for the DF weight w and DF degree of match d is offered below (cf. Table 1 and 2):

DfDifficultyLevel w	
dfDifficultyLevel_Beginner	3
dfDifficultyLevel_Intermediate	4
dfDifficultyLevel_Advanced	5

Table 1: Weights for DF Difficulty Level

DfDifficultyLevel d	AppropriateForBeginners	AppropriateForIntermediates	AppropriateForAdvanced

dfDifficultyLevel_Beginner	0,9	0,05	0,05
dfDifficultyLevel_Intermediate	0,05	0,9	0,05
dfDifficultyLevel_Advanced	0,05	0,05	0,9

Table 2: Degree of Match Score for DF Difficulty Level

3.4.3 Recommender & Personalized Feedback Messages

3.4.3.1 Preliminaries

A **feedback message** is defined as a short text in natural language presented to the user by the LMS and giving the user the possibility to interact with the INTUITEL Engine: reply or cancel feedback messages. More general a feedback message is a **dialogue action**, performed by the Dialogue Manager, which in our case is the Feedback module. Other dialogue actions are updating the database or dropping the current dialogue task.

A **dialogue task** (or dialogue frame) is a state which associates a set of preconditions (or **activation conditions**) based on a given input to a set of **dialogue actions**. The dialogue tasks can be either an inform message, a request, or, user queries. The difference between these tasks is the initiator (inform and request messages are initiated by the system, whereas the query is originated by the user) and the requirement of a user reply (only for a request message is the user required to provide an answer). For the selected didactic scenario, it was considered a priority that the current feedback model should have inform and request messages. Nevertheless, user queries can be easily added as an extension of the current model.

3.4.3.2 Simple dialogue manager for e-Learning feedback

A dialogue task, in a classic management model, passes from *selection* state to *activation*, if certain conditions are met, and then to *execution*. The task maintains the execution state until it expires or it is considered finished. Usually, a task terminates if all the initial requirements are met. In the current implementation, the dialogue management model described in Figure 5 is a simplified version of this approach, since in most of the cases the dialogue task can be considered finished immediately after execution. Moreover, the selection strategy is coupled with the activation condition and only the tasks that can be activated are selected.

Based on an initial requirements analysis, several standard activation conditions for our dialogue tasks were identified. Due to some restrictions in the current system design, only one TUG and LORE message can be active at once. For the dialogue manager this means that only one task can be active at the same time. Additionally, a task remains active (not finished) while the LMS system displays it. According to D1.1, the communication protocol with the LMS includes the possibility of marking any LORE and TUG message as “PAUSED”. Nevertheless, not all the LMS systems inform when the user decides to hide or pause a feedback message; therefore a timeout condition was put in place. The same timeout condition is used to decide if a task is expired. More advanced activation conditions, such as value based activation, can be attached to each individual dialogue task and they are part of the final model.

During the selection phase, two strategies are used: a maximum parameter overlap and a Monte-Carlo selection. The first strategy ranks the list of available dialogue tasks by the overlap between the input parameters and the parameters used in the activation condition. If more than one task has the same rank, the first occurrence is selected. The second selection strategy uses a Monte-Carlo selection algorithm to create a pseudo-random selection. Each task has a selection weight attached, which decreases over time by the inverted popularity of that task. In this way it is ensured that under the same activation conditions, the tasks fulfilling them are selected as uniformly as possible.

In the execution stage, the dialogue action is generated, which can be either a feedback message or a database update.

In conclusion, after analysing all the requirements, several dialogue tasks were identified: inform tasks, complex inform tasks, inform-diagnosis tasks and requests.

Inform Message: These are the simplest tasks available and a series of inputs are associated to the feedback message. The user does not have to react to this message. Therefore the task is considered finished after the feedback is delivered. A diagram of this task (Figure 5) and an example are presented below:

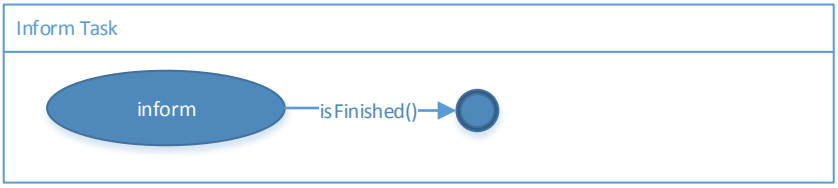


Figure 5: A simple inform task

Inform Message Example:

```
(System) You have completed all the material. Well done!
```

Complex Inform Message: Compared to the previous inform task, this structure allows more flexibility by allowing other tasks to be linked to the current state. The linked tasks have independent activation conditions associated, therefore based on the user feedback on the current state a follow-on message can be selected. In our example and diagram (Figure 6), the complex task is a **yes/no inform**, meaning that the user is asked a simple question to which he can reply yes or no. Depending on his answer, the corresponding inform task is selected. Nevertheless, being a simple inform message, the user is allowed to ignore this question at any point.

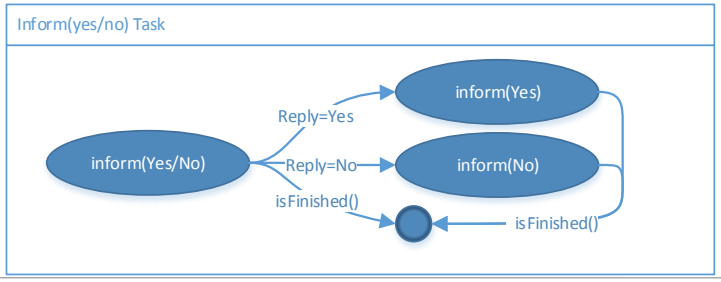


Figure 6: A complex inform task

Complex Inform Message Example:

```
(System) You have spent significantly less time during this session,
         compared to other peers.
(System) Should I offer you more challenging material in future? (yes/no)
(User)   Yes
(System) Your difficulty level is already at the lowest point possible,
         please contact your teacher.
```

Inform followed by a diagnosis task: In all the previously described dialogue tasks, the message activation conditions were not ambiguous. Unfortunately, when dealing with dynamic user models, these conditions can change very easily. In that case, the user profile needs to be diagnosed and

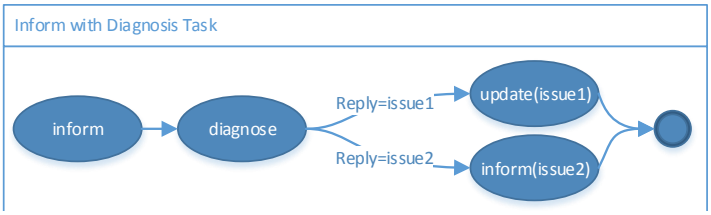


Figure 7: A complex inform feedback followed by a user diagnosis message

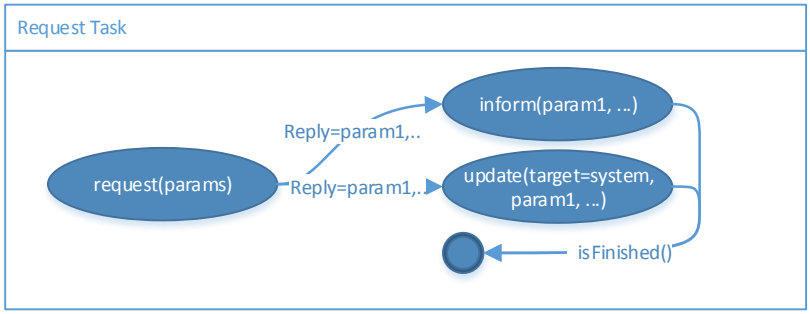
eventually, in case of a resolution, the right action needs to be taken. Figure 7 shows the diagnosis task flow in a simplified scenario. Nevertheless, in real situations, the diagnose task can be more complex and multiple actions can be taken for the same issue. The following example shows one of the feedback messages we have currently modelled for INTUITEL.

Inform Message with Diagnosis Message Example:

```
(System) You have completed half of your learning pathway in the course,
         but your learning pace is very slow.
(System) Are you having trouble with the learning material? (yes/no)
(User)   Yes
(System-action) Update the user database
(System) We decreased your preferred difficulty level. Thank you for the
         feedback.
```

Request Message: In the previous two scenarios, the user was permitted to ignore the feedback message and therefore the dialogue task would be considered finished at any point. In the case of a request, this assumption is no longer valid and the user is required to provide an answer to this feedback messages. Similar to the complex inform messages, the request messages can trigger other inform messages. Moreover, some of the requests messages may trigger update actions. These actions are modelled as abstract as possible, with several existing options, e.g. database updates or

Figure 8: A request dialogue task



informing other components. Figure 8 presents a generic request scenario, followed by a short example.

Request Message Example:

```
(System) In order to personalise your learning material, we require some of
         your profile data. Please fill in the following parameters:
(System) Your age:
(User)   22
 (System-action) Update the user database
```

3.4.3.3 Template-based dialogue management approach

In the previous sections, the dialogue management and task selection was discussed, without a detailed description of how the actual feedback messages are generated. In the current implementation, the dialogue model (i.e. task model) is independent from the NLG template model. This allows more flexibility in terms of language independence and future NLG extensions.

In the template-based dialogue model, the feedback templates are represented as abstract actions, similar to database updates. When a certain dialogue task is selected, if it has a NLG pattern associated, the pattern with all the required parameters, the input options are generated and the message is transformed into html, as required by the TUG format. In the next example, the template requires a runtime parameter **$currentCC**, which is retrieved from the user database.

```
You have completed two thirds of the topic '$currentCC', but your learning
pace is very slow.
```

The next example is more complex and requires a parameter selection depending on the missing input required by the LPM.

```
In order to personalise your learning material, we require some of your
profile data. Please fill in the following parameters:
Your age: (free text input)
Your gender: (male/female/other)
Your preferred language: (English/German/Spanish/Italian/Hebrew)
```

As stated before, the NLG model is language independent. Therefore the previous pattern can be translated into other languages. Currently, the engine supports manually translated NLG models for German and Spanish. Additionally, automatic translation can be easily integrated.

Example, German translation:

```
Sie haben zwei Drittel des Themas '$currentCC' abgeschlossen, aber Ihr
Lerntempo ist sehr langsam.
```

Example, Spanish translation:

```
Ha completado dos tercios del tema '$currentCC', pero su ritmo de
aprendizaje es muy lento.
```

3.4.3.4 NLG-based extensions

In addition to the template-based approach, we offer an NLG-based approach combining single phrases to sentences instead of using predefined template patterns. The principal difference is that the system does not select the NLG pattern, but the associated set of NLG phrases. If there are appropriate alternative phrases for a special NLG phrase, it is thus possible to replace the phrase by a new phrase with the same or a similar meaning. In this manner, one feedback message can have a multitude of possibilities to be realized and recurrence of words can be avoided.

Examples of different possibilities to realize the same feedback message:

```
(1) You have invested more time in the course than your peers.
(2) You have spent more time in this course, compared to your peers.
(3) You have invested more time in the course in comparison to other
    students.
```

The selected phrases are connected to a grammatically correct sentence by using the NLG realizer SimpleNLG, an open source software developed at the University of Edinburgh [33, 34].

Additionally, it is possible to connect two feedback messages. This feature is triggered, if two messages contain the same sentences or multiple identical phrases. In this case, the phrase will be generated only once.

Example: Connecting of two feedback messages (1,2) with the same sentences

```
(1) Your current learning situation is disadvantageous for the course
    material. You can get a faster internet connection. This could increase
    your learning efficiency.
(2) Your current learning situation is disadvantageous for the course
    material. You can proceed in a quieter environment. This could increase
    your learning efficiency.

(output) Your current learning situation is disadvantageous for the course
    material. You can proceed in a quieter environment and get a faster
    internet connection. This could increase your learning efficiency.
```

Other feedback messages can be connected by using conjunction (*but, additionally, etc.*). To avoid wrong or inadequate connections, appropriate connections were defined beforehand and are selected during runtime. Currently, the NLG-based feedback system supports NLG models for English and German.

4 Testing procedures

The feasibility of the overall INTUITEL approach has been validated in form of an extended integration test. A complete INTUITEL system was set up for this purpose, i.e. all components relevant for the creation of learning recommendations and feedback were integrated, including WP05 components as well as the LPM, the LMS Moodle, and the communication layer framework. This also comprised the provision of a semi-authentic test course (real learning material, but not a full course) and its annotation in a proof-of-concept form.

Early testing and debugging of the whole system made it possible to identify problems that would first show up during runtime and which would be hard to identify in purely mocked testing scenarios. Since the INTUITEL processing chain is a sequential system with asynchronous components developed by different partners, special emphasis was put on the API specifications.

A dual testing strategy, i.e. authentic and formalized testing scenarios, were used to test multiple aspects from different perspectives, thus increasing the test coverage.

Summing up it can be stated, that all the tests have been successfully passed. Thus, not only the functionality of the WP05 components has been evaluated, but the whole system, so that first authentic learning recommendations and feedback could be given even before the tests with artificial and real learners (cf. WP12).

Test	Date	Status
Integration Test of INTUITEL System	September 2014	Passed

4.1 Evaluation of WP5 Functionality

The INTUITEL Engine is responsible for obtaining tutorship guidelines based on a model of the learning status of the student and the learner's conversation with the system.

The main components involved in testing the procedure of elaboration of the recommendations are:

- LPM: In charge of elaborating a model of the cognitive and environmental status of the student. This component is simulated for testing the rest of the components.
- QB: In charge of creating and executing the reasoning that results on a tutorship decision.
- RR: In charge of turning the results into natural language artefacts and eventually starting a dialog with the student by means of TUG messages. The output of this component is targeted mainly to the student logged-in in the LMS.
- User Database: Serves as repository of the historic data about the learner's past behaviour and temporal results of the computations. Currently it is implemented as a MySQL server shared by the LPM and RR.

- CL: Interconnection component that routes the messages between the former components. The architecture is event-driven; hence all the communications are asynchronous from the internal point of view.

The testing procedure described in this section is aimed to test the functional behaviour of the tutorship from a pedagogical point of view. The pedagogic experts have developed a set of learning scenarios to be fed into the Engine and a set of expected TUG and LORE messages that are checked after each scenario execution.

The scenarios are formulated as following:

1) An initial state of the system represented as a snapshot of the contents of the User Database.
2) A sequence of LSO documents and User Database updates that simulate the output of the LPM in response of the user's interactions.
3) A set of conditions to be checked against the responses of the INTUITEL Engine in each step of the sequence mentioned in point 2.

Being the system asynchronous by design, the test apparatus needs to be implemented carefully to allow the interception and simulation of the parts that are actually not under testing: LMS and LPM; the rest of the system, that is, the INTUITEL Engine, including all its components, is installed in an dedicated server, together with the Basic Communication Layer as well as the User Database (MySQL server).

The testing framework will use the Communication Library to setup the REST listeners and to communicate with the Communication Layer. The Basic Communication Layer needs to be configured with the actual endpoints of the QB, RR and the simulated URLs defined for the simulated LMS and LPM. This way the messages targeted to the LMS will reach the mocked endpoints of the test framework and the dialog can be simulated by emitting pre-defined messages as response.

Tests have been implemented in Java Language using JUnit framework and a Java-based Domain Specific Language based on the philosophy of the framework Hamcrest that allows expressing declaratively a set of constraints and predicates. These language constructs allow the rapid development of complex conditions and sequences of actions in a test unit.

For example, implementing a communication step in a test case would be expressed as:

```
given().contentType("application/xml")
       .and()
       .body(readFile("UC_01/01_tug_probe.xml"))
       .when()
       .post(tugservicepoint)
              .then()
              .log().body()
              .and()
              .assertThat().body(isXMLSimilarTo(expectedTugResponse));
```

The asynchronous behaviour of the system imposes an asynchronous support for the testing framework. To do this, a set of predicates has been developed to start service points, receive messages and simulate the desired behaviour of the LMS and LPM components.

In the next example a dialog between the RR and the LMS is defined and tested without the need of implementing REST listeners or complex mockups:

```
with(dialog)
       .whenReceives().type("TUG").and().mId(mId).respondWith(expectedTugResponse)
       .whenReceives().type("LORE").and().mId(mId)
                                    .and().uId("john").respondWith("ERROR")
       .then()
              .launchLater(futureTUGmessage,2000)
              .then().dothis().until(10000).waitForMId(mId) // wait for asynchronous
response and check validity
              .then()
              // check conditions
                     .log().body(false)
                     .and()
                     .assertThat().header("type", "TUG")
                     .and()
                     .body(isXMLEquivalentTo(tugMessage));
```

4.2 Evaluation of the Reasoning Back End

One advantage of using a reasoning broker framework in the context of INTUITEL is that it can select the most suitable reasoner(s) depending on the provided input ontology and reasoning request. These differences could be ontological expressiveness, size of the ontologies (in terms of the number of classes or individuals), complexity of the query, etc. [32]. For instance, it might be the case, that for Cognitive Content Models with a high branching factor in the learning pathways, a reasoning engine is more efficient than another one, compared to Cognitive Content Models with low branching factor.

However, in the course of development of the various ontologies in INTUITEL, it turned out that in order to describe KO selection criteria thoroughly, class expressions are required that make use of OWL 2 language features which are not supported by most reasoners. In our experiments, we identified the HermiT reasoned as the only suitable reasoning engine being capable of processing the INTUITEL ontologies.

The advantage of the reasoning broker being able to choose between different reasoned types thus cannot be demonstrated. However, there are still advantages that justify the utilization of the reasoning broker framework:

- Load balancing.

  The HERAKLES Reasoning Broker is ideally configured to utilize various reasoners on different physical or virtual machines, such that each reasoning engine can work independently without being slowed down by other processes on the same machine. This architecture allows for providing different reasoning machines in a cloud infrastructure, which are managed centrally by the HERAKLES Client. Thus, the reasoning broker architecture allows

for the processing of a large number of parallel reasoning requests, and thus serving multiple learners in parallel.

- Flexibility regarding changing background knowledge.

  The fact that the current modelling of KO criteria and pathways can only be processed by a single reasoner does not mean that this will always remain stable in the future. Being flexible with respect to changing background knowledge is one of the key advantages of a knowledge-based system such as the INTUITEL Engine. It might be the case that certain ontology axioms or class expressions might change because of redefinitions or based on the results of long term evaluations. If such a modification might lead to other reasoners being able to process the ontologies more efficiently, the reasoning broker framework will select those reasoners automatically without additional configuration or implementation overhead.

- Flexibility regarding changing reasoner availability.

  Efficient ontology reasoning is a major and active field of research in the knowledge representation community. Existing reasoners are continuously extended, and new reasoners, based on new efficient algorithms are introduced on a regular basis. In case a new or extended reasoner will be available in the future, which is capable of processing the INTUITEL ontologies more efficiently, this reasoner can be employed easily within the existing INTUITEL infrastructure. (It only has to be made available on one or several HERAKLES Reasoner Server machines and registered at the server itself.

The architecture chosen for embedding reasoning engines to the Query Builder via the HERAKLES Reasoning Broker demonstrates high flexibility and load balancing capabilities. This paves the way for making the INTUITEL a robust and scalable Back End for INTUITEL-enabled LMSs.

4.3 Validation of the ranking function

We perform experiments to validate the correctness of the implementation of the ranking module (described in Section 3.4.2).

The evaluation of the utility function involves two main steps:

1. The calculation of the **utility weights** w_k that represent the relative importance of the DF to the overall utility.
   - The relative importance of each DF has been pre-defined by the Didactic Experts and is in the range of [1 – 10].
2. The definition of the **scores** s_{jk} associated with each attribute.
   - The degree of match is approximated with a Gaussian function where a non-linear discounting is used to account for a non-perfect fit. The values range between 0 and 1, with values close to 1 for the best fit.

Validation Test

The following algorithm has been used to validate the correctness of the Ranking module.

Algorithm:

1. Random Choice of a User Setting with features $df_1 \ldots df_{13}$
2. Define space of **best fit** LOs for the given Setting with $df_1 \ldots df_{13}$
3. Compute the best in the fit as a reference score
4. Define a transformation to generate a new LO space with different values with one alternation at a time
5. Calculate the Ranking Score

Distance, d Number of alternations	Resulting Score Value
0	12
1	10
2	9
3	7
4	5
5	5

Figure 9: Computation of Scores for the Validation Test

The experiments confirm (see Fig. 9) that the Ranking Module works as expected: The scores fall off when the number of alternations d increases.

4.4 Evaluation of Feedback Messages

The evaluation of the appropriateness of feedback messages can only be assessed by human users and, in our case, the real learners. Nevertheless, during the development stage the correctness and the integrity of the feedback model was automatically tested and a survey to assess the appropriateness of the feedback messages was conducted.

4.4.1 Testing the correctness and integrity of the feedback model

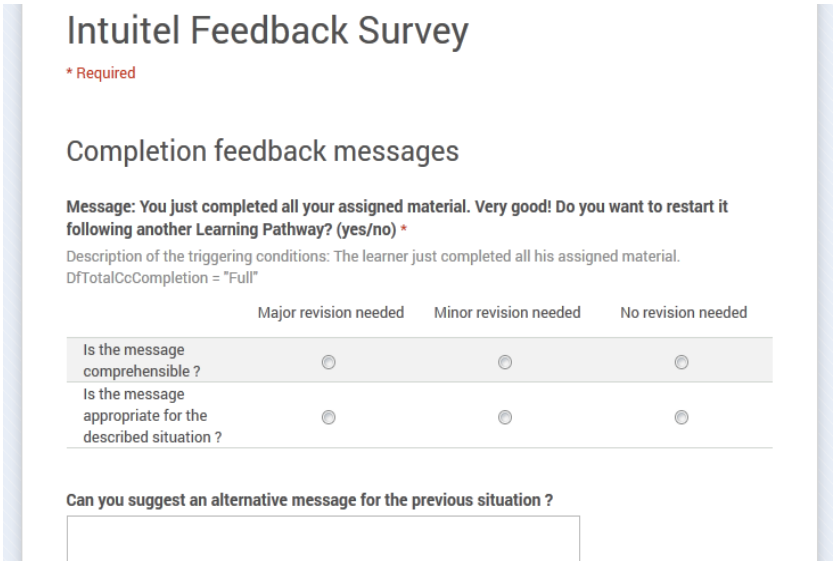
The correctness and the integrity of the feedback model are tested in two steps, using custom JUnit tests. Most of the models are described in multiple XML files, therefore the consistency and integrity of the model has to be tested. This operation is done offline and the model validation is disabled at runtime for speed purpose. The following steps are performed in order to test the integrity of the model:

1. Start and final states are validated. Furthermore, isolated states are identified and considered design errors. At least one start and final state has to be available for a valid model.

2. For any dialogue state, for all the available languages, the consistency of the NLG model is also checked for all the feedback messages. Any inconsistencies across the different NLG models are reported as errors.
3. The validity of input parameters is also tested and any invalid data is reported. Moreover, unused parameters are also checked during this stage.

Testing the compatibility and integrity of the feedback model is easier than assessing the correctness. Similarly to the previous tests, a series of JUnit tests are created to check if the right messages are triggered for a combination of input parameters. Such tests were designed to cover almost all the possible input parameters to make sure that the right feedback messages are triggered.

4.4.2 Assessing the appropriateness of the feedback messages


Intuitel Feedback Survey

* Required

Completion feedback messages

Message: You just completed all your assigned material. Very good! Do you want to restart it following another Learning Pathway? (yes/no) *

Description of the triggering conditions: The learner just completed all his assigned material. DfTotalCcCompletion = "Full"

	Major revision needed	Minor revision needed	No revision needed
Is the message comprehensible ?			
Is the message appropriate for the described situation ?			

Can you suggest an alternative message for the previous situation ?

Figure 10: The Feedback model pre-validation survey

As stated before, the appropriateness of the feedback messages needs to be assessed by human potential learners. Since testing this requires a full working system, an early pre-validation survey was conducted using a Google doc form (see Figure 10).

The survey was carried out internally within the INTUITEL consortium and with research staff from University of Reading, UK. The results of this survey were very positive and only three messages were considered inappropriate to the given learning situation:

1. *You invest more time in the course than your peers. Keep up the good work!* Some of the responders considered that if a learner invests more time with the course it would not necessarily be an improvement. This is a limitation of the current design and it was decided that this message has a lower priority than the other messages.

2. *Your current Micro/Macro Learning Pathway is not set. Please select one of the following options: (option).* This message suggests that the learner is familiar with the INTUITEL terminology. The resolution to this problem is to replace all the unfamiliar INTUITEL terminology with more user friendly options.
3. *You have completed the first third of topic '$currentCC', but your learning pace is very slow.* Some of the responders did not agree with the "first third" formula and preferred the more technical phrase (33%). Others agreed with it. It was concluded that this is a matter of personal preference and it is therefore not addressed by the current prototype.

5 Setting up the system

To get the INTUITEL system running, at least one Windows machine is required. Technically, it is irrelevant if this is the same machine used for the LMS. How many different machines are used is a matter of desired performance. The more learners are expected to work in parallel, the more servers are recommended. Practical experience as an indication for required hardware could not yet be collected.

In the following sections, instructions are given how the individual INTUITEL components need to be set up. In general, a set of (programming) frameworks and programs are required:

- Microsoft .Net version 3.5 (or higher)[10]
- Oracle Java Runtime Environment 7 (or higher)[11]
- MySQL Database, Community Edition, latest version available[12]
- SVN[13] source control client (e.g. Tortoise SVN[14] for Windows) , latest version available
- Apache Ant build tool, latest version available[15]
- *(optional)* Maven build tool, latest version available[16]

The .Net framework is required for the CL. The INTUITEL back-end components are written in JAVA. Running them on a Linux machine should thus be unproblematic. If an INTUITEL-enabled LMS requires any of the above mentioned or additional frameworks is dependent on the LMS, since their underlying technologies differ.

[10] Microsoft .Net runtime: http://www.microsoft.com/
[11] Oracle Java Runtime Environment 1.7: http://www.oracle.com/
[12] MySQL Community Edition: http://www.mysql.com/products/community/
[13] Apache SVN: http://subversion.apache.org/
[14] Tortoise SVN client for Windows: http://tortoisesvn.net/
[15] Apache Ant build toolkit: http://ant.apache.org/
[16] Apache Maven build toolkit: http://maven.apache.org/

After setting up all components, is it required to first start the Communication Layer instance and the LMS (if this requires starting some kind of active component – e.g. in the case of ILIAS). All other components, i.e. LPM, QB, RR and SLOM repository can be executed in arbitrary order.

5.1 LMS

The INTUITEL core described in this document can be integrated with any INTUITEL-enabled LMS. To make such integration possible, the LMS needs to implement the USE/TUG/LORE interface as described by D1.1.

Of course the LMS set-up depend on the actual target LMS. As a proof of concept, and also an actual feasibility demonstration, into the INTUITEL project we extend and integrate five LMSs, using heterogeneous technologies:

- ILIAS: Php and Java, extended by HSKA
- Moodle: Php, extended by UVA
- Clix Mobile: NodeJs, extended by IMC
- Crayons: Java, extended by Fraunhofer IOSB
- Exact LCMS: C#, extended by ELS

At the time of writing, all the five LMSs have been extended with an INTUITEL USE/TUG/LORE interface as described in D1.2. Having now also the INTUITEL core modules up and running, the consortium is elaborating test data and proceeding with integration. Improvements and adjustments of the LMSs are scheduled for the third year of the INTUITEL project in WP8 and WP9.

5.2 Communication Layer

5.2.1 Basic Communication Layer

5.2.1.1 Components

The current implementation is composed of 2 projects:

- CommLayerServer (CL Server): The core project in charge of processing and routing messages.
- MockIntuitelServices (Mock CL Services): Auxiliary project implementing all INTUITEL services to test purposes (LMS, LPM, RR, Query Builder and LsoRepository). They return mock responses and implement all the INTUITEL interfaces.

5.2.1.2 Deployment

The Basic CL Server is published and can be downloaded from the following url: http://95.211.177.222/Intuitel/IntuiBasicCL.zip. The steps for its deployment are following:

- Unzip the content of the file downloaded.

- Configuration of the CommLayerServer:
  - The main executable is “IntuitelConsoleHost.exe”.
  - The configuration is distributed in 2 files:
    - IntuitelConsoleHost.exe.config: Stores the connection database, log configuration, the main service address and advanced service configuration.
    - IntuitelRoutes.config: Stores all the messages types, connected services and routes.

5.2.1.3 Configuring the Basic CL Server

5.2.1.3.1 IntuitelConsoleHost.exe.config

The file shows the following properties:

- In *AppSettings* Section:
  - DEFAULT_SERVICEHOST_ADDRESS: Endpoint to listen to new requests, i.e. http://localhost:8181/CommunicationManagerService
  - CONFIG_FILE_PATH: Full path pointing the “IntuitelRoutes.config”. However, this property doesn’t need to be changed.
- In *log4net* section:
  - file: It should contain the path to the log file.
- In *System.ServiceModel* section:
  - “BaseAddress”: It contains the same value as DEFAULT_SERVICEHOST_ADDRESS, i.e. http://localhost:8181/CommunicationManagerService
  - httpGetUrl: It’s used to publish the wsdl file of each service, i.e. http://localhost:8181/CommunicationManagerService

5.2.1.3.2 IntuitelRoutes.config

This configuration file shows the following properties:

- *MessageTypes* section: this is properly configured to recognize all the possible messages developed within INTUITEL, therefore it is not necessary to perform any change on it. As explanation, MessageTypes section contains a list of IntuitelMessageType elements where:
  - Name: Identifies the message type. (Mandatory)
  - XPathRecognition: XPath to match in the analysis process. (Mandatory)
  - XPathMessageIdValue: XPath which returns the Id element of the message to be able to send the response when a message with a rId attribute is processed.

- XPathResponsesEndpointValue: XPath which returns the uri of the sender in the message to be able to send the response when a message with an rId attribute is processed.

- *Components* Section: This section is configured by default with the mock services implemented, therefore some changes are required. It is composed of a component list. Every component represents an INTUITEL service with the following characteristics (see Figure 11 as example):
  - Name: Name of the component. (Mandatory).
  - EndPoint: Base endpoint of the component. (Mandatory)
  - ServiceName: Identifies the Interface implemented by the component. (Mandatory for Microsoft Windows Communication Framework communications).
  - ConnectionType: One value among “Rest”, “Soap” or “Wcf”. (Mandatory)
  - MethodList: List of methods provided by the component (Instances of IntuitelComponentService class). Every Method contains the following properties:
    - Name: The method’s name. (Mandatory).
    - ContentType: The content type sent in the request. This is only applicable for REST communication being then the values either Json or Xml. Xml is the default one.
    - ReturnType: Not currently used. The system detects automatically the return type in all the connection types.
    - ParamList: List of the parameters needed to call this web service method. Every parameter has:
      - Name: Name of the parameter. (Mandatory).
      - Type: Type of Param. Mandatory for Rest. It should be one of:
        - RequestBody: Send the parameter value as a payload. If this parameter is set, its value will be sent as the body of the request. Only one RequestBody Parameter is accepted – the first one. The name of the parameter will be used as the Content-Type header for the request.
        - HttpHeader: Adds the parameter as a HTTP header that is sent along with the request. The Header Name is the name of the Parameter and the Header value is the Value.
        - Cookie: Adds the parameter to the list of cookies that are sent along with the Request. The Cookie Name is the name of the Parameter and the Value is the Value (.ToString) passed in.
        - UrlSegment: this ParameterType replaces placeholder values in the RequestUrl:

- Method: "health/*{entity}*/status"
- When the request is executed, the system will try to match any *{placeholder}* with a parameter of that name (without the {}) and replace it with the value. So the above code results in “health/s2/status” being the endpoint of the url of the callable method of the component.
  - ValueIsConstant: Means if the value to be sent is a constant value provided in the Value property.
  - Value: Value to send. The special value is “[message]”, to send the content of the request to route.

```
<IntuitelComponent>
  <Name>QueryBuilder</Name>
  <EndPoint>http://localhost:8091/QueryBuilderService/rest</EndPoint>
  <ServiceName>IQueryBuilderService</ServiceName>
  <ConnectionType>Rest</ConnectionType>
  <MethodList>
    <IntuitelComponentService>
      <Name>LsoNotification</Name>
      <ContentType>Xml</ContentType>
      <ParamList>
        <IntuitelServiceMethodParameter>
          <Name>application/xml</Name>
          <Type>RequestBody</Type>
          <Value>[message]</Value>
          <ValueIsConstant>false</ValueIsConstant>
        </IntuitelServiceMethodParameter>
      </ParamList>
      <ReturnType>Xml</ReturnType>
    </IntuitelComponentService>
  </MethodList>
</IntuitelComponent>
```

Figure 11: Sample XML code for an INTUITEL Component

- In the components’ list, it is needed to change the EndPoint of the Service to implement and the paramList. For a normal REST XML communication with payload, this is the correct configuration of a component. The system always makes POST calls.
- *Routes* section: This section is configured currently with the correct routes to the services. Every IntuitelRoute has the following properties:
  - SourceMessageType: The message type which activates this route. If several routes have the same sourceMessageType, the system will select the first in the list. (Mandatory).
  - DestinationComponentName: Component to send the received message. (Mandatory).
  - MethodNameToBeCall: Method of the destination component to call. (Mandatory).
  - FixedResponse: If the caller needs a response in its request, the system will send this as a response, apart from send the message to the destination component.

5.2.1.4 Execution of the Basic Communication Layer

To initialize the server, the "IntuitelConsoleHost.exe" has to be executed. The host requires *administrator* rights to be executed, so if the logged user is not an administrator (or belongs to the administrator's group), he has to execute the host as administrator (right click and "Run As Administrator"). Figure 12 shows the output of the Basic CL server once it is executed.

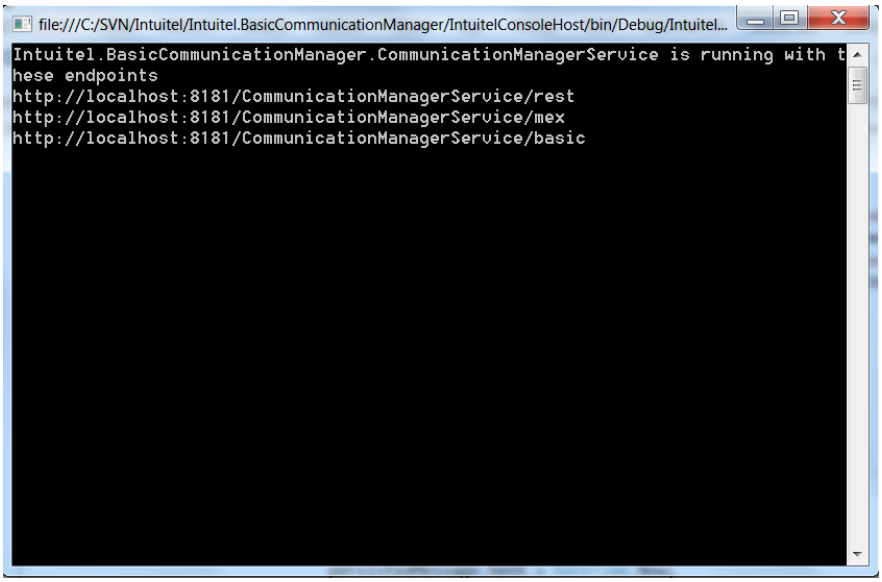


Figure 12: Basic CL Server output

5.2.2 Advanced Communication Layer

5.2.2.1 Installation Advanced Communication Layer

As already mentioned in D 3.3, the Advanced Communication Layer, in advance AdvCL, is composed by TSB and an XMPP Server. XMPP messages are XML messages sent through a XMPP server. TSB is configured to recognize INTUITEL messages through XPath expressions. To be able to install and configure the AdvCL, a TSB server properly configured and a XMPP server are needed. An option for XMPP Server is OpenFire[17]. To setup and configure TSB, it is necessary to obtain a license from TIE Kinetix and the installation and configuration is performed by TIE's consultants.

5.2.2.2 Configuring Advanced and Basic Communication Layer

To setup the AdvCL it is necessary to create the following Document Structures in TSB:

- LearnerUpdatePullRequest
  - XPath Recognition: /intuilms:INTUITEL/intuilms:Learner[@pushType="false" and not(*)] | /INTUITEL/Learner[@pushType="false" and not(*)] |

[17] OpenFire link: http://www.igniterealtime.org/downloads/download-landing.jsp?file=openfire/openfire_3_9_3.exe

/intuilms:INTUITEL/intuilms:Learner[not(@pushType) and not(*)] | /INTUITEL/Learner[not(@pushType) and not(*)]

- XPath Message Id Value: /intuilms:INTUITEL/intuilms:Learner/@mId | /INTUITEL/Learner/@mId
- XPath Responses Endpoint Value: /intuilms:INTUITEL/@lmsUri | /INTUITEL/@lmsUri

- LearnerUpdatePushRequest
  - XPath Recognition: /intuilms:INTUITEL/intuilms:Learner[@pushType="true" and not(*)] | /INTUITEL/Learner[@pushType="true" and not(*)]
  - XPath Message Id Value: /intuilms:INTUITEL/intuilms:Learner/@mId | /INTUITEL/Learner/@mId
  - XPath Responses Endpoint Value: /intuilms:INTUITEL/@lmsUri | /INTUITEL/@lmsUri
- LearnerUpdateResponse
  - XPathRecognition:/intuilpm:INTUITEL/intuilpm:Learner[count(*)>0] | /INTUITEL/Learner[count(*)>0]
- LearnerUpdatePollingRequest
  - XPathRecognition: /intuilpm:INTUITEL/intuilpm:Learners[@mId and not(/intuilpm:INTUITEL/intuilpm:Learners/intuilpm:Learner)] | /INTUITEL/Learners[@mId and not(/INTUITEL/Learners/Learner)]
- LearnerUpdatePollingResponse
  - XPath Recognition: /intuilms:INTUITEL/intuilms:Learners[@mId]/intuilms:Learner | /INTUITEL/Learners[@mId]/Learner
- UseEnvRequest
  - XPath Recognition: /intuilpm:INTUITEL/intuilpm:UseEnv/intuilpm:Data[not(@value)] | /INTUITEL/UseEnv/Data[not(@value)]
  - XPath Responses Message Id Value: /intuilpm:INTUITEL/intuilpm:UseEnv/@rId | /INTUITEL/UseEnv/@rId
- UsePerfRequest
  - XPath Recognition: /intuilpm:INTUITEL/intuilpm:UsePerf/intuilpm:LoPerf[not(*)] | /INTUITEL/UsePerf/LoPerf[not(*)]
  - XPath Responses Message Id Value: /intuilpm:INTUITEL/intuilpm:UsePerf/@rId | /INTUITEL/UsePerf/@rId
- UseResponse

- XPathRecognition: /intuilms:INTUITEL/intuilms:UsePerf/intuilms:LoPerf/intuilms:Score | /intuilms:INTUITEL/intuilms:UseEnv/intuilms:Data[@value] | /INTUITEL/UsePerf/LoPerf/Score | /INTUITEL/UseEnv/Data[@value]

- TugRequest
  - XPathRecognition: /intuilpm:INTUITEL/intuilpm:Tug[not(@retVal)] | /INTUITEL/Tug[not(@retVal) and not(/INTUITEL/Tug/Data)]
  - XPath Responses Message Id Value: /intuilpm:INTUITEL/intuilpm:Tug/@rId | /INTUITEL/Tug/@rId
- TugResponse:
  - XPathRecognition: /intuilms:INTUITEL/intuilms:Tug[@retVal] | /INTUITEL/Tug[@retVal]
- TugDelayedResponse:
  - XPathRecognition: /intuilms:INTUITEL/intuilms:Tug/intuilms:Data | /INTUITEL/Tug/Data
- LoreRequest
  - XPathRecognition: /intuilpm:INTUITEL/intuilpm:Lore[not(@retVal)] | /INTUITEL/Lore[not(@retVal)]
  - XPath Responses Message Id Value: /intuilpm:INTUITEL/intuilpm:Lore/@rId | /INTUITEL/Lore/@rId
- LoreResponse
  - XPathRecognition: /intuilms:INTUITEL/intuilms:Lore[@retVal] | /INTUITEL/Lore[@retVal]
- LmsProfileRequest
  - XPathRecognition: /intuilpm:INTUITEL/intuilpm:LmsProfile[not(*)] | /INTUITEL/LmsProfile[not(*)]
- LmsProfileResponse
  - XPathRecognition: /intuilms:INTUITEL/intuilms:LmsProfile[count(*)>0] | /INTUITEL/LmsProfile[count(*)>0]
- LoMappingRequest
  - XPathRecognition: /intuilpm:INTUITEL/intuilpm:LoMapping | /INTUITEL/LoMapping[count(*)<2]
- LoMappingResponse
  - XPathRecognition: /intuilms:INTUITEL/intuilms:LoMapping | /INTUITEL/LoMapping[count(*)>1]
- AuthRequest

- XPathRecognition: /intuilpm:INTUITEL/intuilpm:Authentication/intuilpm:Pass | /INTUITEL/Authentication/Pass

- AuthResponse
  - XPathRecognition: /intuilms:INTUITEL/intuilms:Authentication/@status | /INTUITEL/Authentication/@status

5.2.2.3 Using the Advanced Communication Library

To ease the interchangeability between both versions of the CL, the same interfaces of the Basic CL are kept; however, the AdvCL internally makes XMPP calls with the correspondent messages in a correct format. The only new interface is the *OpenAdvanced* method, which is needed to open connections to the XMPP server.

```
OpenAdvanced(String username, String password, String xmppServer, int
                    xmppServerPort, String tsbUsername);
```

This method (callable by the LpmServiceConnection or LmsServiceConnection), opens a connection to the XMPP Server with these parameters:

- Username: Username to login in the server.
- Password: password to login in the server.
- XmppServer: address of the XMPP Server.
- XmppServerPort: port to access the XMPP Server.
- TsbUsername: username of the advanced communication layer managed by TSB through XMPP network.

5.3 User Database

5.3.1 Requirements

1. MySQL Community Edition or better[18], latest version available

5.3.2 MySQL Install

There are various MySQL options to install, depending on the platform. Some MySQL toolkits for Windows or Linux come bundled with a management interface:

1) XAMPP[19] bundle, has the XAMPP console installed
2) EasyPHP[20] comes packed with the well-known PhpMyAdmin interface
3) MySQL Workbench[21] is the default management tool developed by the Oracle Corporation.

[18] MySQL Community Edition: http://www.mysql.com/products/community/
[19] XAMPP: https://www.apachefriends.org/
[20] EasyPHP: http://www.easyphp.org/
[21] MySQL Workbench: http://www.mysql.com/products/workbench/

5.3.3 Setup the database

Note: For now, the hibernate.cfg.xml has to be placed inside the jar file. The easiest option to setup the database is to create a default username (**intuitel**) and password (**ursus**).

1) Start a local MySQL database engine (e.g. via XAMPP or EasyPHP).
2) Create DB-user for localhost **intuitel** with the password **ursus**. Alternatively, a different username and password can be chosen, but the User Database library needs to be rebuilt.
3) Create a new **intuitel** database for the new DB-user. Please make sure that the newly created user (**intuitel**) has connection rights from **localhost** on the **intuitel** database.
4) Execute the provided *intuitelDB.sql* script on the **intuitel** database. This could be also executed as a MySQL import.

Note: Please be sure that the default storage engine for the created database is MyISAM, as it is faster than other options. InnoDB should be supported, but MyISAM is recommended.

5.4 SLOM repository

The SLOM repository is the persistence layer that stores the course enriched data in order to make it available for later reuse. Different applications have been used to implement this functionality starting with the most important one: MongoDB database.

MongoDB is an open-source noSQL database and document oriented. This database has been chosen due to the following reasons:

- It is an agile database that allows schemas to change quickly as applications evolve, while still providing the functionality from traditional databases, such as secondary indexes, a full query language and strict consistency.
- MongoDB is built for scalability, performance and high availability, scaling from single server deployments to large, complex multi-site architectures.
- By leveraging in-memory computing, MongoDB provides high performance for both read and write tasks.
- MongoDB's native replication and automated failover enable enterprise-grade reliability and operational flexibility.

On the other hand, with the objective of ease the interaction with the SLOM repository, an administration tool called RockMongo (PHP Administrator) has been used.

Finally, closely linked to the SLOM repository, an intermediate service component, the SLOM repository Communication Service, has been developed in order to make possible the communication with the INTUITEL's CL component, and therefore making the SLOM repository reachable from other INTUITEL's components.

MongoDB installation is a requirement of the SLOM Repository. As a third party software, installation guidelines for the latest available version can be found at the provider's web site[22]. The steps taken

[22] http://www.mongodb.org/

to install MongoDB and RockMongo in the prototype setup are included in the document as Appendix.

5.4.1 SLOM repository Collections

After installing the RockMongo tool and using the user interface it is possible to create manually all the different collections (tables on a typical SQL database) that will store all the necessary information related to a course. This process can be accelerated using MongoDB command shell and a JavaScript file that will create these collections automatically.

Next is shown the script file “mdbsetup.js” used to create the MongoDB collections needed for the SLOM Repository:

```
print("Creating database collections...");
db.createCollection("lmsRepository");
db.createCollection("course");
db.createCollection("macroLP");
db.createCollection("conceptContainer");
db.createCollection("ccConnector");
db.createCollection("microLP");
db.createCollection("knowledgeObject");
db.createCollection("koConnector");
db.createCollection("file");
print("Database collections created.");
print("Creating database user owner...");
db.createUser({user:"intuiteluser", pwd:"intuitelpass ", roles: [ "readWrite", "dbOwner" ]});
```

This script file can be executed using the system command prompt:

```
C:\mongodb\bin\mongo.exe localhost:27017/intuitel  mdbsetup.js
```

The parameter “localhost:27017/intuitel” indicates that mongo program has to connect with the “intuitel” database even if that database does not exists. MongoDB does not provide any command to create “database“. In fact, MongoDB will create it on the fly, right after the first “createOperation” is called. The result of the execution of this script can be seen in Figure 13:

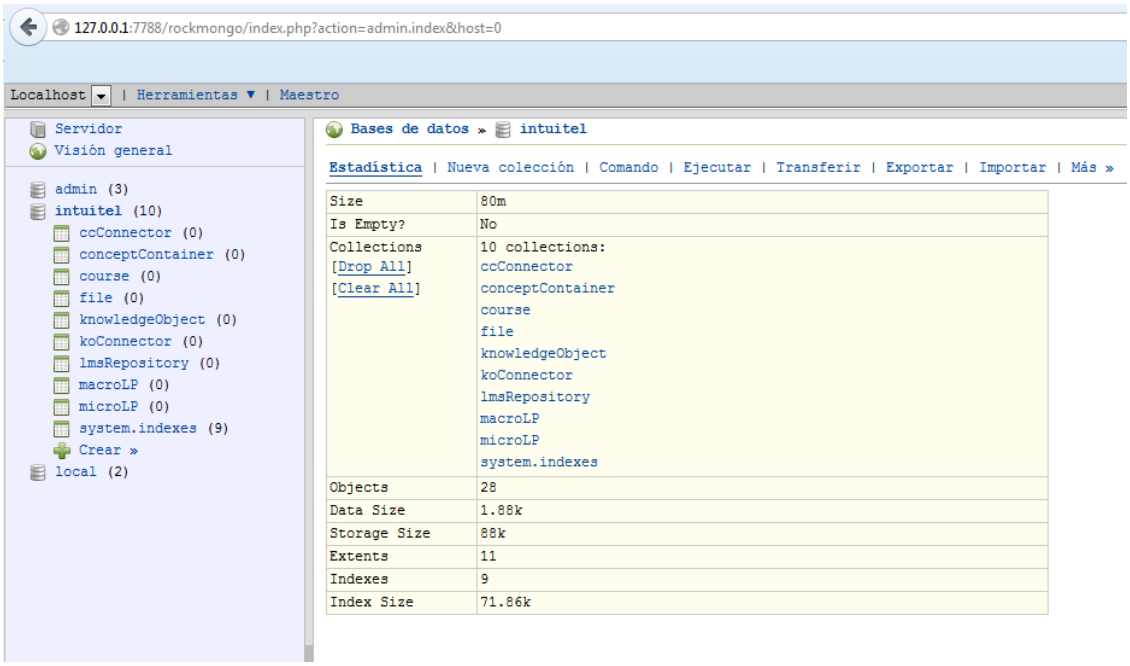


Figure 13: MongoDB SLOM Repository database in RockMongo

Finally, in MongoDB the early creation of the "fields" included on each collection is not needed. These fields will be implicitly generated once the first "insert" operation is performed on a collection. Also the primary key *_id* will be automatically added if *_id* field is not specified.

5.4.2 SLOM repository Communication

SLOM repository is used as a persistence layer for the INTUITEL Editor's tool but it also shares information with other components in the INTUITEL Platform through the CL component.

The INTUITEL Editor uses the Java MongoDB Driver to access the database. The last version of the driver can be downloaded from the maven central repository[23] and the last API documentation can be found in MongoDB's web site[24].

This driver is also used by the SLOM repository to allow the information exchange with the CL Component regardless which version of the CL is used. This application service has two main operations:

- Systematically monitors the SLOM repository looking for course changes and informs of that changes to other components inside INTUITEL system.
- Allows read operations on the SLOM repository exposing course information to other components.

[23] MongoDB driver in the Maven Central Repository: http://central.maven.org/maven2/org/mongodb/mongo-java-driver/

[24] MongoDB's API: http://api.mongodb.org/java/current/index.html

NB: The writing functionality has been disabled from the SLOM repository public access since the INTUITEL Editor is the only tool allowed to change the data in the SLOM repository.

5.5 Learning Progress Model

The LPM is provided as a JAR file, which can be executed from command line. Apart from JRE7, no additional installations are required.

Configuration of the LPM takes place via command line arguments. Using the "-help" argument prints a list of all available arguments including descriptions for more detailed information.

To start the LPM using the basic CL, two parameters need to be specified:

- "lpmAddress": The URL which the LPM itself should use.

  e.g.: http://localhost:8182/Lpm

- "clAddress": The URL of the CL's REST endpoint.

  e.g.: http://localhost:8181/CommunicationManagerService/rest

These addresses need to be matched with the configuration of the basic CL (cf.: section 5.2.2.2).

To start the LPM using the advanced CL, five parameters need to be specified:

- "xmppUser": The username to be used for connecting to the advanced CL.

  e.g.: lpm

- "xmppPsw": The password for above specified user.

  e.g.: lpm

- "xmppServerAdr": IP-address of the advanced CL.

  e.g.: 95.211.177.222

- "xmppServerPort": Port of the advanced CL.

  e.g.: 5222

- "tsbUser": User for connecting to the TSB part of the advanced CL.

  e.g.: tsb@95.211.177.222

Further configuration of the advanced CL is not necessary.

Since the LPM is also the main agent for storing and data about the learning process, a MySql database needs to be provided. The database software used for this should be irrelevant. For the internal tests and development, XAMPP was used (https://www.apachefriends.org/index.html). The LPM expects a user called "intuitel" with a password "ursus" to connect to the database. The required tables can be created using the SQL-script which is being delivered with the LPM jar-file. Since the LPM and the Recommendation Rewriter both access this database (cf. Section 5.3), this consequently has to be matched with the RR configuration.

5.6 Query Builder and Reasoning Broker

5.6.1 Building the Query Builder

The build process of the Query Builder component is managed by Apache Maven. In order to build an executable JAR-file including all dependencies, execute the appropriate Maven command within the Query Builder directory containing the Maven configuration file (WP05/Query Builder/pom.xml).

```
_> mvn clean package
```

The package `Query Builder.X.X.X-jar-with-dependencies.jar` will then be available in `WP05/Query Builder/target`.

Note: The two Communication Layer libraries `eu.intuitel.CommLayer.CommLibrary` and `eu.intuitel.CommLayer.Commons` must be available in the local Maven repository in order to successfully build the Query Builder package.

5.6.2 Building the HERAKLES Reasoning Server

The build process of the HERAKLES Reasoning Server is the same as for the Query Builder. An executable JAR-file including all dependencies is created by executing the appropriate Maven command within the HERAKLES Reasoning Server directory containing the Maven configuration file.

```
_> mvn clean package
```

5.6.3 Configuring the Query Builder

Configuration of the Query Builder component is done via the configuration file `WP05/Query Builder/setup.properties`.

The following configuration options are available:

If the Advanced communication layer is used, the following options are relevant:

- `username`
- `password`
- `xmppServer`
- `xmppServerPort`
- `tsbUsername`

If the Basic communication layer is use, the following options are relevant:

- `uri` (e.g. `http://localhost/querybuilder`)
- `intuitelGateway` (e.g. `http://localhost:8181/CommunicationManagerService/rest`)

Whether the Basic or the Advanced communication layer is to be used by the Query Builder is determined by the following option:

- `commLayerType` (valid configuration: "`basic`" or "`advanced`")

Moreover it can be configured, whether to use the communication layer or some mockup layer. The latter is only required for testing/debugging purposes and usually does not have to changed:

- `connectionType` (valid config: "`commLayer`" or "`mockupLayer`")

The number of concurrent workers can be determined by the following option:

- `numberWorkers`

The HERAKLES Reasoning Servers to be used can be specified via the following option, where the different server addresses are provided as a semicolon-separated list of server URLs:

- `heraklesURLs`

Before executing the Query Builder, ensure that "`setup.properties`" is available in the same directory as the JAR-file created during the build phase.

5.6.4 Configuring HERAKLES Reasoning Servers

In order to set up a HERAKLES Reasoning Server an according "`setup.properties`" file has to be provided in the same directory as the JAR-file created for the HERAKLES Reasoning Server during the build phase.

The properties has to contain a set of key-value pairs, where the key denotes a reasoner ID and the value the qualified Java class name of the factory-class used for creating an instance of the reasoner. This allows for using arbitrary OWL reasoners via the HERAKLES server by simply adding all classes and dependencies in the classpath and referring to the factory-class in the properties file.

5.6.5 Installing and Executing the Query Builder

Use the following command to execute the Query Builder component without showing any further logging:

```
_> java -cp querybuilder-X.X.X-jar-with-dependencies.jar
     eu.intuitel.querybuilder.service.QueryBuilderService
```

In order to show more detailed logging for debugging purposes, provide a `log4j2.xml` configuration file:

```
_> java -cp querybuilder-X.X.X-jar-with-dependencies.jar
     -Dlog4j.configurationFile=./log4j2.xml
     eu.intuitel.querybuilder.service.QueryBuilderService
```

You may use a preconfigured configuration file `WP05/QueryBuilder/log4j2.xml` which switches debug logging on.

5.6.6 Installing and Executing a HERAKLES Reasoning Server

Use the following command to execute the HERAKLES Reasoning Server, either on the same machine as the Query Builder, or on a different one:

```
_> java -jar rest-jettyserver-X.X.X-jar-with-dependencies.jar -[port]
```

where `[port]` specifies the port at which the HERAKLES Reasoning Server is accessible. The default value port, in case no port is specified, is 8080.

5.7 Recommendation Rewriter

5.7.1 Compile/Release the service

In the downloaded source code folder, the Recommender Rewriter folder, execute the following command:

```
_> ant release
```

This will release all the needed jars and script in the **prod/** folder of the service**.**

5.7.2 Configure the service

5.7.2.1 Main configuration

```
#
# Creator: INTUITEL Consortium, www.intuitel.eu
# Contributor: Ovidiu Şerban, ISR Laboratory, University of Reading, UK, http://www.isr.reading.ac.uk/
# Version: 1.00
# Date: 2014/7/3
# Copyright: Copyright (C) INTUITEL Consortium
# License: The license that applies to this file has not yet been determined
#

dialogueModel=intuitel-mockup/inform-dialogue-model.xml,intuitel-mockup/request-dialogue-model.xml
rankingModel=ranking/Ranking.properties
fillerModel=dbFiller.xml
#Recommender Rewrite Service
workerQueueSize=10
connectionType=mockup
dbType=mockup
#Communication Layer Properties for advanced layer
username=rr
password=rr
xmppServer=95.211.177.222
xmppServerPort=5222
tsbUsername=tsb@95.211.177.222
#Communication Layer Properties for the basic layer
basicUri=localhost:8032/rr
```

```
basicGateway=localhost:8181/CommunicationManagerService/rest
```

Code 1: The Recommender Rewrite Properties file (recommendedRewrite.properties)

Properties:

1. dialogueModel – points to the feedback models to be loaded by the service. Usually it is left unchanged.
2. rankingModel – points to the ranking model configuration file. Usually it is left unchanged.
3. fillerModel - points to the DB filler model. Usually it is left unchanged.
4. workerQueue – this configures the processing queue size for the feedback module. This should be left unchanged for testing purpose. Usually it is left unchanged.
5. connectionType – refers to the connection type used with the communication layer. If set to **mockup**, the service does not use the actual communication library. To use it, this setting should be changed to **commLayer**.
6. dbType – refers to the database connection type. If set to **mockup** the component uses a local cache. **Hibernate** uses a database connection.
7. advanced communication layer properties – to be setup with the credentials provided communication layer setup
8. basic communication layer properties
    a. basicUri – a unique port and name has to be given and represents the rest interface for this basic component (e.g. http://localhost:8032/rr)
    b. basicGateway – the rest interface of the communication manager (http://localhost:8181/CommunicationManagerService/rest)

Note: Only one communication layer setup is accepted. If the basic layer is configured, the advanced options are disabled by default.

5.7.2.2 Ranking properties

The ranking strategy is configured manually in ranking\Ranking.properties:

```
#
# Creator: INTUITEL Consortium, www.intuitel.eu
# Contributor: Andrea Zielinski, IOSB Fraunhofer, Germany,
http://www.iosb.fraunhofer.de/
# Version: 1.00
# Date: 2014/7/3
# Copyright: Copyright (C) INTUITEL Consortium
# License: The license that applies to this file has not yet been
determined

# Properties file for Ranking
# User Settings
RecommendationStrategy = HardCriteria
RecommendationStrategy = SoftCriteria
# Default Settings
Axiom = Recommendation Axiom 1.1
```

```
Axiom = Recommendation Axiom 1.2
Axiom = Recommendation Axiom 1.3
Axiom = Recommendation Axiom 1.4
```

Code 2: The ranking properties file (Ranking.properties)

The Recommendation Strategy and Axiom has to be configured according to the testing scenario. The axioms specify if the recommended LOs are direct successors of the current node (Recommendation Axiom 1.1), all direct successors of the current KO (Recommendation Axiom 1.2), all predecessors of the current KO (Recommendation Axiom 1.3) or direct successors of the previous KO (Recommendation Axiom 1.4) (cf. Section 3.3.2.2).

Moreover, the ranking values can be modified by the tutor, i.e. in ranking\data\dfWeights and ranking\data\dfDegreeOfMatch.

5.7.3 Running the service

To run the service, execute in a Windows command line console:

```
_> start-RR.bat
```

5.7.4 Extending the feedback model

5.7.4.1 Translating a feedback model

The NLG feedback model is independent from the dialogue model, therefore in case a new language is supported (i.e. Italian), two new NLG models have to be created: *inform-nlg-model.ita.xml* and *request-nlg-model.ita.xml.* The first corresponds to inform messages, while the second is the request model. The English model can be used as a template for the new models. The next step is to import the new NLG models into the corresponding dialogue model (inform or request). This is done by adding the following line in the dialogue models:

```
<importNLG language="ita">inform-nlg-model.ita.xml</importNLG>
```

Once imported, these models can be translated manually or semi-automatically with any existing tool.

Note: The language codes used by this module are in **ISO 639-3** (three letters) format[25]. These codes are similar to the **ISO 639-1/2** (two letters) format. The only major change in the naming convention, for the languages supported by the project, is done for Spanish: **ISO 639-3** is spa and **ISO 639-1/2** is es.

[25] Wikipedia ISO 639-3 Article: http://en.wikipedia.org/wiki/ISO_639-3

5.7.4.2 Adding new feedback messages

The current NLG feedback model is structured into nlgPattern tags, which contain multiple patterns linked to the current structure. In the next example, the pattern that needs to be extended has been identified and another pattern has been created and will be added to the model.

```
<NlgModel>
    <nlgPattern id="nlgPatternId">
        <nlgPattern>
            <pattern>You just completed all your assigned material.
            </pattern>
        </nlgPattern>
        <nlgAnswers>
            <answer paramName="invalidateLpSelection" type="single"
                    description="Do you want to restart it following
another Learning Pathway?">
                <option value="Y">yes</option>
                <option value="N">no</option>
            </answer>
        </nlgAnswers>
    </nlgPattern>
```

The new pattern:

```
<pattern>Very good! You just completed all your assigned material.
</pattern>
```

Additionally, existing answer templates can be also extended in a similar way to the patterns.

6 Use of different reasoning engines

6.1 Definition of the task

The INTUITEL Engine as a knowledge-based system is flexible regarding changing didactical theories, learner models, and course material. Once set up, the system can incorporate e.g. new pedagogical findings, newly available sensor data regarding learner states, or new learning material for existing courses, without the need to change the processing logic of the INTUITEL Engine.

While gaining maximum flexibility, this design requires robust and efficient utilization of reasoning engines for processing the involved ontologies. The HERKLES Reasoning Broker used in the INTUITEL Engine provides a framework for robust and efficient utilization of external reasoning engines. Thereby, reasoning engines can be used in parallel and according to their strengths regarding the reasoning tasks in INTUITEL.

6.2 Step by step instructions

The HERAKLES Reasoning Broker is used by the Query Builder just as any particular reasoned, since it is implementing the `OWLReasoner` interface provided by the OWL API. This integration is done in the Query Builder and does not require any additional configuration.

In order to make use of different reasoning engines via the HERAKLES Reasoning Broker, the following steps need to be taken:

1. Provide one or several physical or virtual machines, on which reasoners can be hosted.
2. Install reasoners on those machines. Detailed instructions on how to obtain reasoners and use reasoners should be provided by the reasoned developers.
3. On each machine, install the HERAKLES Reasoning Server according to the instructions provided in Section 5.6.4.
4. Configure the HERAKLES Reasoning Server according to the instructions provided in Section 5.6.4 making known to the Server, which reasoners are available. Note: The Reasoner Factory Classes need to be available in the Java Classpath.
5. Set the URLs of the HERAKLES Reasoner Servers in the Query Builder configuration file according to the instructions provided in Section 5.6.4. This makes the HERAKLES Reasoning Servers known to the HERAKLES Client which is integrated in the Query Builder.

7 Discussion / Conclusion

7.1 General

In this deliverable, we discussed the strength and the weaknesses of a pure ontology-based approach as the motivation for the design of a hybrid recommender system that combines a reasoning and ranking approach. While pure knowledge-based approaches are able to identify appropriate Learning Objects that meet the learner's profile (personalized recommendation), they do not provide a ranking on recommended Learning Objects. We propose a metric for LO selection based on a Utility Function. The rationale behind this is that for a given user profile an order can be deduced based on multiple criteria that originate from the setting of Didactical Factors, metadata of the Learning Objects including their completeness state, current KO and selected learning pathway.

We propose a system architecture, where ranking takes place after reasoning in a post-processing step. Our hybrid system design efficiently handles multiple requests by different learners, since queries are distributed over multiple engines using the Query Builder via the HERAKLES Reasoning Broker. Complex conjunctive queries are avoided and results are computed over disjoints and resolved in the Ranking Phase. This also has the advantage that intermediate results of reasoning can be reused and recombined at a later stage. Moreover, in the case that no Learning Object completely satisfies all conjuncts of a given complex conjunctive class expression, the learning resources which fulfil most of the constraints can be determined easily using the ranking-based approach. Furthermore, soft constraints are supported in our framework. While many approaches face difficulties to give a recommendation when the dataset is sparse, our solution is to rank all Learning Objects and calculate the closest best-fitting Learning Object to the learner. One major contribution of this deliverable is a novel formalization to handling learning pathways as structured sequences in OWL which could also be used for dynamic courseware generation.

Last but not least, we have made a technical achievement in developing software code that has been successfully tested on a sample test scenario of an e-Learning course and integrated into the LMS Moodle. The User guide is an integral part of the effort and constitutes the basis on which LMS developers can semantically enhance their system based on the INTUITEL framework.

8 References

[1] Aleman-Meza, B., Halaschek-Weiner, C., Arpinar, I.B., Ramakrishnan, C., Sheth, A.P.: Ranking complex relationships on the semantic web. Internet Computing, IEEE 9(3), 37-44 (2005)

[2] Alian, M., Jabri, R.: A Shortest Adaptive Learning Path in eLearning Systems: Mathematical View. Journal of American Science 5(6), 32-42 (2009)

[3] Anyanwu, K., Maduko, A., Sheth, A.: Semrank: ranking complex relationship search results on the semantic web. In: Proc. of the 14th Intl. Conf. on World Wide Web. pp. 117-127. ACM (2005)

[4] Aroyo, L., Dolog, P., Houben, G.J., Kravcik, M., Naeve, A., Nilsson, M., Wild, F.: Interoperability in personalized adaptive learning. Educational Technology & Society 9(2), 4-18 (2006)

[5] Barros, H., Silva, A., Costa, E., Bittencourt, I.I., Holanda, O., Sales, L.: Steps, techniques, and technologies for the development of intelligent applications based on semantic web services: A case study in e-learning systems. Engineering Applications of Artificial Intelligence 24(8), 1355-1367 (2011)

[6] Bock, J., Tserendorj, T., Xu, Y., Wissmann, J., Grimm, S.: A Reasoning Broker Framework for OWL. In: Proc. of the 6th Intl. Workshop on OWL: Experiences and Directions (OWLED). vol. 529. CEUR Workshop Proceedings (2009)

[7] Brusilovsky, P., Mill_an, E.: User models for adaptive hypermedia and adaptive educational systems. In: The adaptive web. pp. 3-53. Springer (2007)

[8] Devedzic, V.: Education and the Semantic Web. Intl. Journal of Artificial Intelligence in Education 14(2), 165-191 (2004)

[9] Dolog, P., Henze, N., Nejdl, W., Sintek, M.: Personalization in distributed elearning environments. In: Proc. of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 170-179. ACM (2004)

[10] Drummond, N., Rector, A.L., Stevens, R., Moulton, G., Horridge, M., Wang, H., Seidenberg, J.: Putting OWL in Order: Patterns for Sequences in OWL. In: Proc. of the Workshop on OWL: Experiences and Directions (OWLED 2006) (2006)

[11] Gaeta, M., Orciuoli, F., Paolozzi, S., Salerno, S.: Ontology Extraction for Knowledge Reuse: The e-Learning Perspective. IEEE Transactions on Systems Man and Cybernetics; Part A - Systems and Humans 41(4), 798-809 (2011)

[12] Gaeta, M., Orciuoli, F., Ritrovato, P.: Advanced ontology management system for personalised e-Learning. Knowledge-Based Systems 22(4), 292-301 (2009)

[13] Gueye, B., Rigaux, P., Spyratos, N.: Taxonomy-based annotation of xml documents: Application to elearning resources. In: Methods and Applications of Artificial Intelligence, pp. 33-42. Springer (2004)

[14] Henze, N., Dolog, P., Nejdl, W.: Reasoning and Ontologies for Personalized ELearning. Educational Technology & Society 7(4), 82-97 (2004)

[15] Henze, N., Nejdl, W.: Logically Characterizing Adaptive Educational Hypermedia Systems. In: Proc. of the Workshop on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2003) (2003)

[16] Janssen, J., Berlanga, A., Vogten, H., Koper, R.: Towards a learning path specification. Intl. Journal of Continuing Engineering Education and Life-Long Learning 18(1), 77-97 (2008)

[17] Karampiperis, P., Sampson, D.G.: Adaptive instructional planning using ontologies. In: ICALT. vol. 4, pp. 126-1 (2004)

[18] Kasneci, G., Suchanek, F.M., Ifrim, G., Ramanath, M., Weikum, G.: Naga: Searching and ranking knowledge. In: 24th Intl. Conf. on Data Engineering, ICDE 2008. pp. 953-962. IEEE (2008)

[19] Keet, C.M.: Ontology engineering with rough concepts and instances. In: Knowledge Engineering and Management by the Masses, pp. 503-513. Springer (2010)

[20] Kerkiri, T., Manitsaris, A., Mavridou, A.: Reputation Metadata for Recommending Personalized e-Learning Resources. In: Proc. of the 2nd Intl.Workshop on Semantic Media Adaptation and Personalization (2007)

[21] Lukasiewicz, T., Straccia, U.: Top-k retrieval in description logic programs under vagueness for the semantic web. In: Scalable Uncertainty Management, pp. 16-30. Springer (2007)

[22] Magliacane, S., Bozzon, A., Della Valle, E.: Effcient execution of top-k sparql queries. In: The Semantic Web{ISWC 2012, pp. 344-360. Springer (2012)

[23] Markellou, P., Mousourouli, I., Spiros, S., Tsakalidis, A.: Using semantic web mining technologies for personalized e-learning experiences. Proc. of the web-based education pp. 461-826 (2005)

[24] Min, W.X., Wei, C., Lei, C.: Research of Ontology-based Adaptive Learning System. In: Proc. of the Intl. Symposium on Computational Intelligence and Design (ISCID '08). vol. 2, pp. 366-370 (2008)

[25] Schockaert, S., De Cock, M., Kerre, E.E.: Reasoning about fuzzy temporal information from the web: towards retrieval of historical events. Soft Computing 14(8), 869-886 (2010)

[26] Shaw, C.J.: System design and architecture of an online, adaptive, and personalized learning platform. Ph.D. thesis, Massachusetts Institute of Technology (2013)

[27] Shen, L.P., Shen, R.M.: Ontology-based learning content recommendation. Intl. Journal of Continuing Engineering Education and Life Long Learning 15(3-6), 308-317 (2005)

[28] Stojanovic, L., Staab, S., Studer, R.: eLearning based on the Semantic Web. In: Proc. of the World Conference on the WWW and Internet (WebNet2001) (2001)

[29] Swertz, C., Schmölz, A., Forstner, A., Heberle, F., Henning, P., Streicher, A., Bargel, B.A., Bock, J., Zander, S.: A Pedagogical Ontology as a Playground in Adaptive Elearning Environments. In: Proc. of the 7th Intl. Workshop on Applications of Semantic Technologies (AST) (2013)

[30] Yu, Z., Nakamura, Y., Jang, S., Kajita, S., Mase, K.: Ontology-Based Semantic Recommendation for Context-Aware E-Learning. In: Proc. of the 4th Intl. Conf. on Ubiquitous Intelligence and Computing. LNCS, vol. 4611. Springer (2007)

[31] Bock, J. (2014). Reasoning Brokerage: New Reasoning Strategies. In Towards the Internet of Services: The THESEUS Research Program (pp. 121-130). Springer International Publishing.

[32] Bock, J., Lösch, U., & Wang, H. (2012, June). Automatic reasoner selection using machine learning. In Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics (p. 23). ACM.

[33] Marcel Bollman (2011). Adapting SimpleNLG to German. To appear in Proceedings of the 13th European Workshop on Natural Language Generation (ENLG-11).

[34] Albert Gatt and Ehud Reiter (2009). SimpleNLG: A realisation engine for practical applications. Proceedings of the 12th European Workshop on Natural Language Generation (ENLG-09).

9 Appendix

Figure 14: Architectural overview

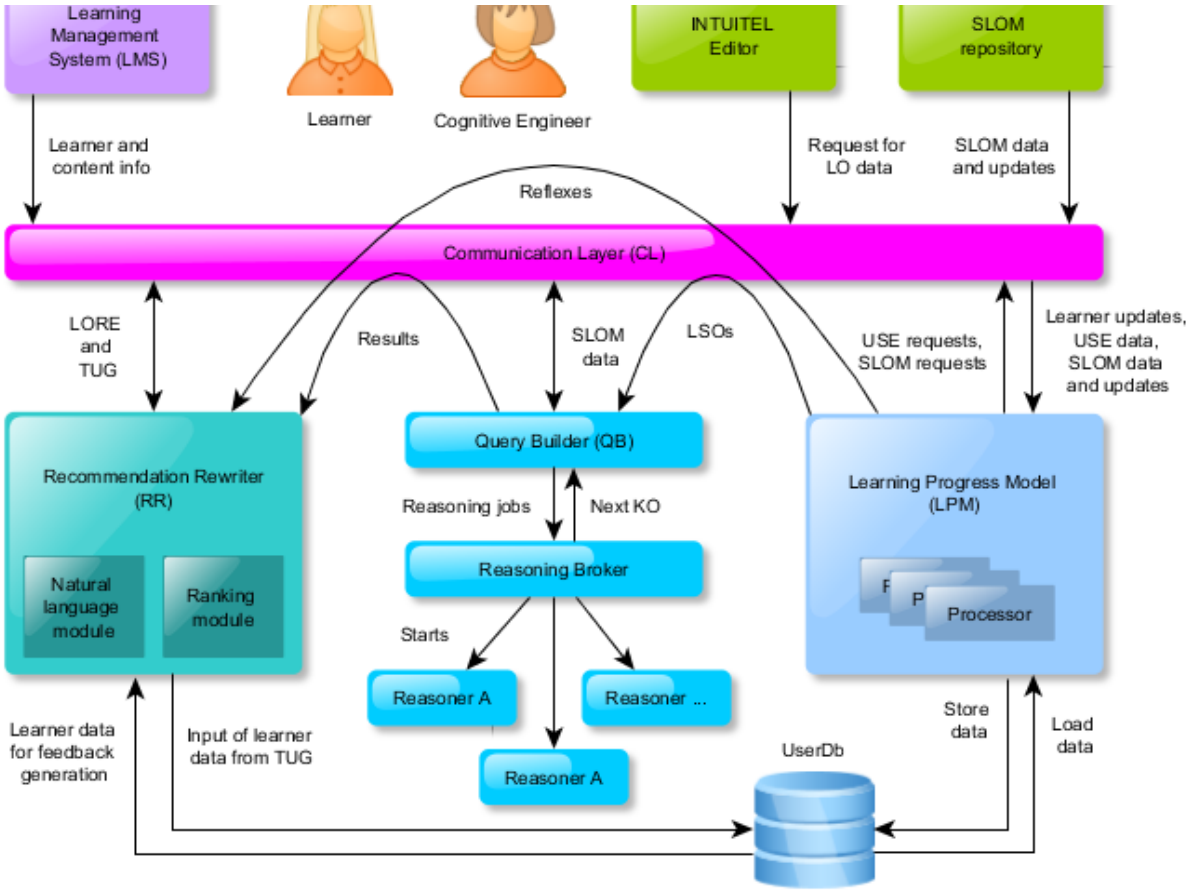


Figure 15: Sequence diagram leading to a first recommendation for a learner

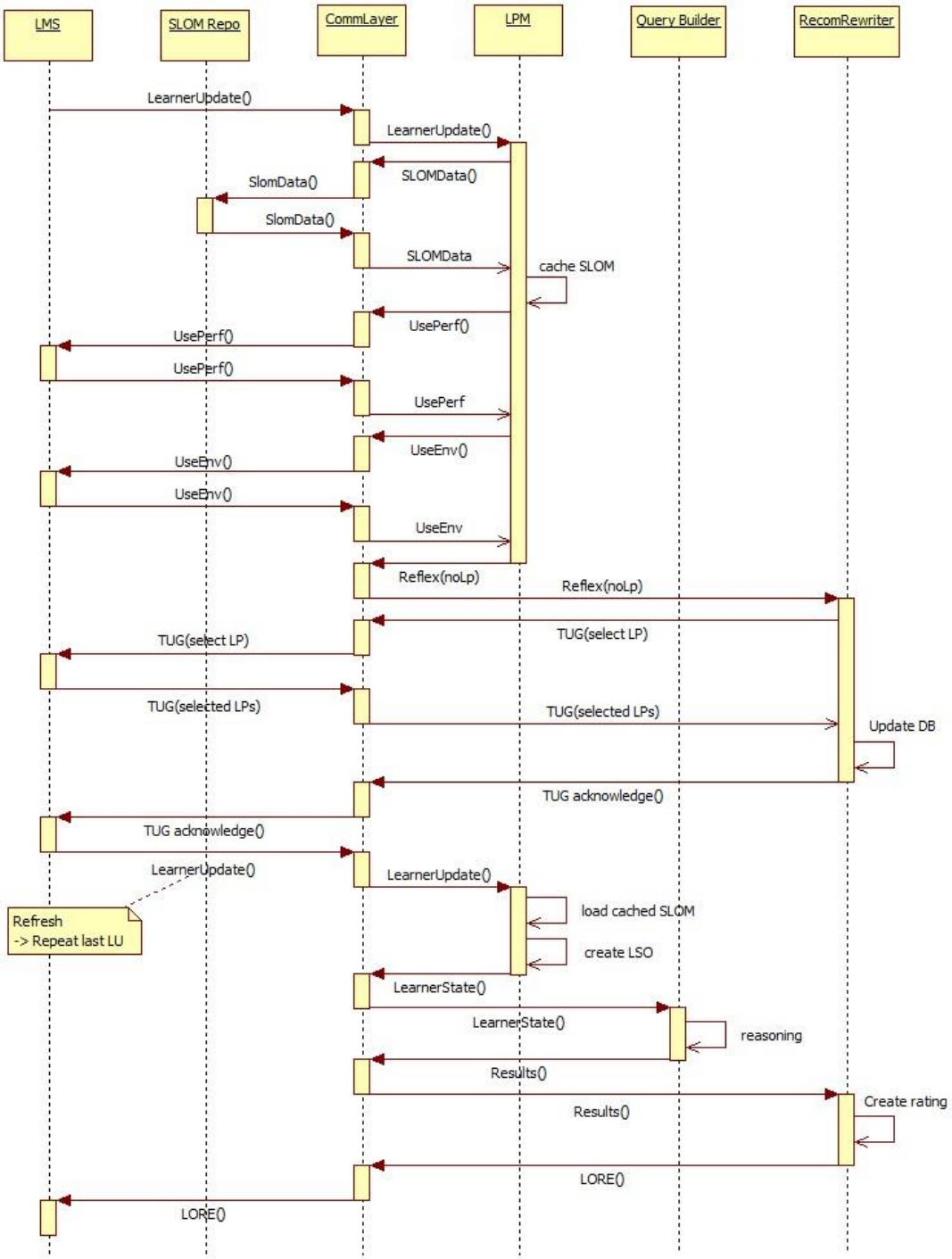


Figure 16: INTUITEL Database Design

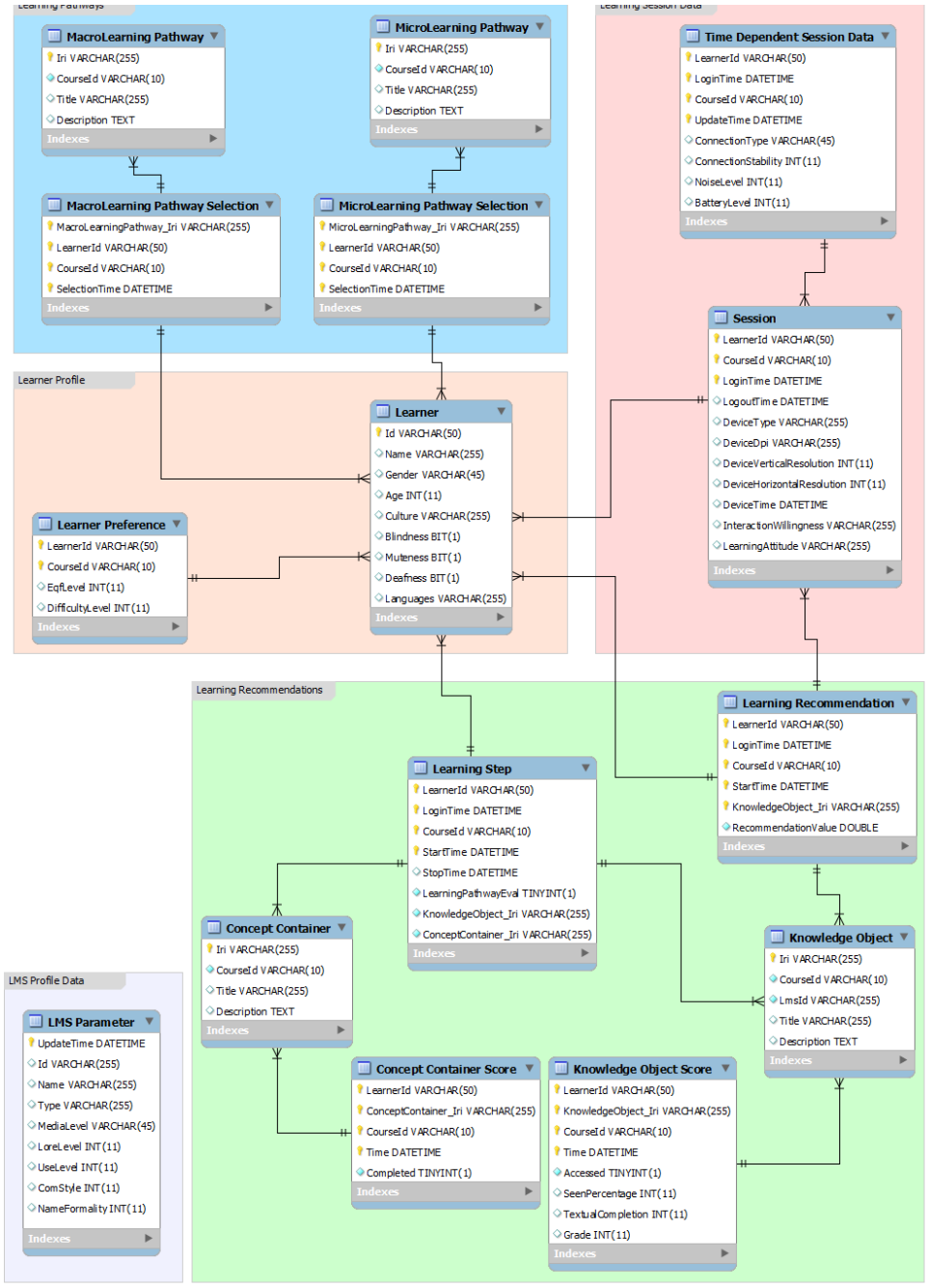


Figure 17: General architecture of the Recommender Rewriter