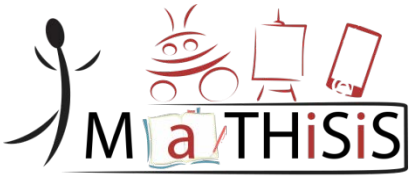

Managing Affective-learning THrough Intelligent atoms and Smart InteractionS

D3.4 The MaTHiSiS Learning Graphs M24

Workpackage	WP3 – Smart Learning Atoms and Graph Tools
Editor(s):	Dorothea TSATSOU, CERTH Alexandros PAPACHRISTOU, CERTH Thomas TECHENE, DXT
Responsible Partner:	CERTH
Quality Reviewers	Evaggelos SPYROU, NSCR Andrew POMAZANSKYI, NG
Status-Version:	Final v1.0
Date:	Project Start Date: 01/01/2016; Duration: 36 months Deliverable Due Date: 31/12/2017 Submission Date: 22/12/2017
EC Distribution:	Report, Public
Abstract:	This deliverable defines and exemplifies the structure for representing both the global as well as the personal model of a learning scenario to the MaTHiSiS learners, namely the MaTHiSiS Learning Graphs (LGs). It further describes the library implementation that handles the creation, access and modification of the Learning Graphs, namely the LG lib, as well as the corresponding Open API implementation used to seamlessly access the features of LG lib throughout the MaTHiSiS platform. This is the second and final iteration of this document, rounding the modelling

This document is issued within the frame and for the purpose of the *MATHISIS* project. This project has received funding from the European Union's Horizon 2020 Programme (H2020-ICT-2015) under Grant Agreement No. 687772

	strategy, the RESTful services that are offered through the MaTHiSiS ecosystem and the front-end interface used to create, view, access and modify Learning Graphs, with the additional report of updates since the first iteration of this document and first version of these components.
Keywords:	Learning Graph, Learning Graph Instance, non-linear educational model, Learning Content Editor, LG lib, LG lib Open API
Related Deliverable(s)	D3.2 The MaTHiSiS Smart Learning Atoms M24 D3.3 The MaTHiSiS Learning Graphs M12 D3.6 Experience Engine M24 D3.9 MaTHiSiS Frontend Components M24 D6.2 The MaTHiSiS Learning Graph Engine M24 D7.3 MaTHiSiS platform, 2nd release D4.5 Multimodal learning analytics D7.3 MaTHiSiS Platform, 2nd release

Document History

Version	Date	Change editors	Changes
0.1	16/11/2017	Thomas TECHENE, DXT	Initial Version of the document
0.2	04/12/2017	Thomas TECHENE, DXT	Contribution section 3
0.3	19/12/2017	Dorothea TSATSOU, CERTH	Version for internal review
0.4.1	20/12/2017	Evaggelos SPYROU, NSCR	Internal review
0.4.2	20/12/2017	Andrew Pomazanskyi, NG	Internal review
0.5	21/12/2017	Alexandros PAPACHRISTOU, CERTH	Integration of review comments, final version of Appendices
0.6	21/12/2017	Dorothea TSATSOU, CERTH	Version for quality review
0.6	22/12/2017	Ana Piñuela, ATOS	Final Quality Review
1.0	22/12/2017		FINAL VERSION TO BE SUBMITTED

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
ASC	Autism Spectrum Case
CGDLC	Career Guidance and Distance Learning Case
ITC	Industrial Training Case
LA	Learning Action
LAM	Learning Action Materialization
LCM	Learning Content Manager
LCS	Learning Content Space
LES	Learning Experience Supervisor
LG	Learning Graph
LGI	Learning Graph Instance
LGE	Learning Graph Engine
MEC	Mainstream Education Case
PA	Platform Agent
PMLDC	Profound and Multiple Learning Disabilities Case
rtm	Runtime (single instance of a MaTHiSiS structure in time)
SLA	Smart Learning Atom
SLAI	Smart Learning Atom Instance
US	User Space

Table 1: Definitions, Acronyms and Abbreviations

Project Description

The MaTHiSiS learning vision is to provide a novel advanced digital ecosystem for vocational training, and special needs and mainstream education for individuals with an intellectual disability (ID), autism and neuro-typical learners in school-based and adult education learning contexts. This ecosystem consists of an integrated platform, along with a set of re-usable learning components with capabilities for: i) adaptive learning, ii) automatic feedback, iii) automatic assessment of learners' progress and behavioural state, iv) affective learning, and v) game-based learning.

In addition to a learning ecosystem capable of responding to a learner's affective state, the MaTHiSiS project will introduce a novel approach to structuring the learning scenario for each educational (sub)domain and each learner. Learning graphs act as a novel educational structural tool. The building materials of these graphs are drawn from a set of Smart Learning Atoms (SLAs) and a set of specific learning goals which will constitute the vertices of these graphs, while relations between SLAs and learning goals constitute the edges of the graphs. SLAs are atomic and complete pieces of knowledge which can be learned and assessed in a single, short-term iteration, targeting certain problems. More than one SLA, working together on the same Learning Graph, will enable individuals to reach their learning and training goals. Learning goals and SLAs will be scoped in collaboration with learners themselves, teachers and trainers in formal and non-formal education contexts (general education, vocational training, lifelong training and specific skills learning).

MaTHiSiS is a 36-month long project co-funded by the European Commission Horizon 2020 Programme (H2020-ICT-2015), under Grant Agreement No. 687772.

Executive Summary

The current document is the deliverable ***D3.4 - The MaTHiSiS Learning Graphs M24*** and it comprises the second and final iteration of the report, which describes one of the outcomes of work package ***WP3 - Smart Learning Atoms and Graph Tools*** and more specifically ***T3.2 - Learning Graphs Implementation***. It is closely related to Deliverable D3.2 – The MaTHiSiS Smart Learning Atoms [4] and its outcomes will be core to the implementation of the MaTHiSiS Learning Graph Engine (LGE) [8].

This document summarizes the logic, objective and definitions presented in the first iteration of this document (D3.3 [5]), in order to stand as a self-sustained documentation of the introduced educational tool. It further describes how Learning Graphs (LG) are integrated and accessed within the MaTHiSiS ecosystem, outlining all relevant updates since the first version of the implementation. Finally, it extends to real-life examples from the MaTHiSiS assisted pilots that outline the main attributes and innovations of LGs. It is a public document, aimed to introduce this educational tool to relevant stakeholders, while it will constitute the main reference document for the integration of LGs in the MaTHiSiS ecosystem.

This deliverable will firstly delve into the definition of the concept of LGs and relevant concepts within the MaTHiSiS ecosystem, and will further detail the corresponding LG representation structures and functionalities implemented in order to create, access and manipulate LGs. The LG-related aspects of the Learning Content Editor (detailed from other points of view in Deliverables D3.2 [4], D3.6 [6] and D3.9 [7]), a human-machine interface developed for the editing of MaTHiSiS concepts, is also presented in this deliverable. These structures, methods and interface have been widely used during the driver and assisted pilots, yielding valuable user feedback and bringing about new technical requirements, on top of the initial implementation reported in the previous iteration of this document (D3.3 [5]), based on which they had all evolved over time.

The **Learning Graph** is a novel educational structural tool introduced in MaTHiSiS, which enables *non-linear execution of a learning scenario while fostering personalised and adaptive learning*. It consists of *learning objective components* (i.e. learning goals and SLAs) *and weighted relations between them*. The LG will guide the process of organising and deploying a learning scenario and will lead to the achievement of an educator's teaching/training objectives. In a nutshell, LGs consist of all the interconnected components/concepts pertaining to what-to-learn per learning scenario during the educational process.

The *Smart Learning Atoms (SLAs)* are atomic and complete pieces of learner knowledge, competencies and/or skills. SLAs essentially comprise *primordial learning goals, constituents of more advanced learning goals, which cannot be further reduced to more primitive notions*. In a nutshell, they consist of the simplest of concepts pertaining to *what-to-learn* during the educational process which partake but are not restricted to a particular learning scenario (or LG). In part, *learning goals* also describe learners' skills or knowledge, which are however more compound and span over a larger comprehensive learning objective. Accordingly, learning goals consist of the *broader competences the learners need to acquire in order to achieve specific learning objectives*. In a nutshell, they consist of composite concepts pertaining to *what-to-learn* per learning scenario and thus are specific to a particular learning scenario (or LG). Although more than one scenarios/LGs may include conceptually similar goals, learning goals are defined as those competences that are inherently too complex to bear independent self-sustainable attributes, as opposed to SLAs.

The core Learning Graphs educational attributes are:

- *Ubiquity* that refers to the capacity of Learning Graphs to holistically represent domain models of skills/knowledge/competences of concrete learning scenarios, and thus enable

training of target learning objectives for all relevant learners under any possible educational setting, context and technological capacities.

- *Individuality* refers to the capacity of the LGs to be instantiated to strictly personal structures (Learning Graph Instances) that carry each learner's personal level of achievement in all pieces of knowledge/skill/competence that the LG represents.
- *Non-linearity* pertains to the ability of the learning objective components of an LG to be trained non-sequentially, based solely on the learner-specific dependencies during a learning experience. In principle, during the MaTHiSiS learning experience, specific learning activities are deployed for a selected SLA, thus training the particular SLA's skill through the designated activity which consequently trains the learner over the more composite learning goals

The *LG library* incorporates all the methods and functionalities required to create, access and modify the LG, LGI (Learning Graph Instance) and runtime LGI data structures. It is implemented as a Java library which is embedded to the Open API. The Open API used to edit, update and delete LGs.

The LGs intervene in several places of the MaTHiSiS Front-end:

- In the Learning Content Manager (LCE), all the LGs created by MaTHiSiS users, stored in the Learning Graphs Repository (LGR), can be browsed, viewed and edited.
- The LCE, where tutors can create and publish new LGs, as well as edit and update existing ones.
- In the Learning Experience Supervisor, LGs can be seen by tutors and learners in their Learning Experiences.

1. Introduction

1.1 Purpose of the document

The MaTHiSiS learning approach relies on modelling learning scenarios in Learning Graphs, i.e., graph structures that comprise the domain model of a wide learning objective, relevant to all learners undertaking it, and encapsulate the specific knowledge, skills and/or competences to acquire during the learning process, interconnected towards the achievement of the overall objective.

Furthermore, the Learning Graph modelling strategy is able to capture the learner-specific model instances that represent the progress/uptake each specific learner, which takes part in the learning process, over the overall learning scenario. Progress is measured based on both personalised metrics, captured historically during the learner's experience with the MaTHiSiS ecosystem, as well as through the learner's affective response and performance over targeted learning activities (cf. D6.2 [8] for more details).

This document is a public technical report, intended to describe the structure underneath the core MaTHiSiS learning approach, i.e. the Learning Graphs. "Task 3.2 Learning Graph Implementation" partners, namely CERTH and DXT, have been involved in the definition of this concept. CERTH is responsible for the implementation of the LG library and corresponding Open API used to represent, access and modify learning scenarios through unpersonalised LG structures as well as to create, access and modify personalised learner-specific instances of these structures. DXT is responsible for developing the interface, employed by the end users of the platform, in order to compose, retrieve and modify unpersonalised LGs.

1.2 Structure of the document

This document contains the key sections detailed below:

- **Section 1: Introduction**
  This section clarifies the purpose and the structure of this deliverable and its context.
- **Section 2: Learning Graphs**
  The second section details the definition of the Learning Graphs (LGs), why they have been introduced in the MaTHiSiS context and related concepts that LGs have a dependent relation to. Furthermore, the methodology behind the integration of this concept in the MaTHiSiS system is described. It concludes with concrete examples of Learning Graphs to better illustrate the rationale behind them and their use in the MaTHiSiS driver and assisted pilots.
- **Section 3: Learning Graph library implementation details**
  The third section describes how Learning Graphs have been integrated in MaTHiSiS technically. The functionalities associated with the creation and manipulation of LGs are described, along with the description of the RESTful Open API used to edit, update and delete LGs. Finally, the front-end subcomponents related to Learning Graphs are detailed, to complete the integration implantation of this concept throughout all layers of the platform.
- **Section 4: Conclusion**
  This section presents the conclusions of the document and future enhancements in the work related to ***T3.2 - Learning Graphs Implementation***.

2. Learning Graphs

This section unfolds all concepts, definitions and details related with the MaTHiSiS core educational objective representation tool, namely the Learning Graphs (LGs).

2.1 Objectives and definitions

This section presents and elucidates the processes involved in a Learning Experience in the MaTHiSiS ecosystem, in which Learning Graphs constitute the pivotal representational structure. A **Learning Experience** "refers to any interaction, course, program, or other experience in which learning takes place, whether it occurs in traditional academic settings (schools, classrooms) or non-traditional settings (outside-of-school locations, outdoor environments), or whether it includes traditional educational interactions (students learning from teachers and professors) or non-traditional interactions (students learning through games and interactive software applications)"[11]. In the context of MaTHiSiS, it *refers to learning/training that takes place based on a specific MaTHiSiS-induced learning scenario*.

As aforementioned in the previous section, Learning Graphs consist the domain model of the particular competences targeted within a particular learning scenario. A **Learning Scenario** is "an a priori description of a learning situation, independently of the underlying pedagogical approach. It describes its organization with the goal of ensuring the appropriation of a precise set of knowledge, competences or skills" [12]. In other words, in MaTHiSiS, *a learning scenario ensures the appropriation of the knowledge, competence and/or skills encapsulated in a specific Learning Graph*.

A **Learning Graph (LG)** consists of *learning objective components* (i.e. learning goals and SLAs) *and weighted relations between them*. The LG will guide the process of organising and deploying a learning scenario and will lead to the achievement of an educator's teaching/training objectives. In a nutshell, LGs consist of all the interconnected components/concepts pertaining to **what-to-learn** per learning scenario during the educational process.

As described in Deliverable 3.2 *The MaTHiSiS Smart Learning Atoms* [4], **Smart Learning Atoms (SLAs)** are atomic and complete pieces of learner knowledge, competencies and/or skills. SLAs essentially comprise *primordial learning goals, constituents of more advanced learning goals, which cannot be further reduced to more primitive notions*. In a nutshell, they consist of the **simplest of concepts** pertaining to *what-to-learn* during the educational process which partake but are not restricted to a particular learning scenario (or LG), as described in D3.2 Section 2.3, for two of their most prominent attributes are atomicity/self-sustainability and re-usability.

In part, **learning goals** also describe learners' skills or knowledge, which are however more compound and span over a larger comprehensive learning objective. Accordingly, learning goals consist of the *broader competences the learners need to acquire in order to achieve specific learning objectives*. In a nutshell, they consist of **composite concepts** pertaining to *what-to-learn* per learning scenario and thus are specific to a particular learning scenario (or LG). Although more than one scenarios/LGs may include conceptually similar goals, learning goals are defined as those competences that are inherently too complex to bear independent self-sustainable attributes, as opposed to SLAs.

The holistic, domain-oriented, modelling of learning scenarios in Learning Graphs, enables them to be **omnipresent** in all situational contexts, able to train the learning objective through a plurality of agents and learning environments. The most important innovation imbued in LGs is the ability to allow for a **highly adaptive, non-linear discretisation** of the learning process. The relational organisation of the learning objective components into interconnected atomic and composite units enables the learning experience to alternate upon mastering individual learning objective components in order to master the overall learning objective of each LG. Alternation between

constituents adopts each learner's own learning style, mood and situational circumstances, eliminating the constraint of following a cascading, group-based continuum, as in the case of traditional learning experiences.

2.2 Methodology and dependencies

In order to facilitate the common representation of learning objectives and their adaptability to different learner specifications, LGs will take up two forms in the MaTHiSiS learning setting, with an added structure used for analysing the learning progress.

1. **Unpersonalised, core LGs** comprise the universal domain model of the MaTHiSiS learning scenaria into a connected graph $G = (V, E)$ of interconnected learning objective components (SLAs and learning goals). The interconnections, represented by directed edges E) of these components (vertices V) form a hierarchy between the simpler components to the more composite ones, thus denoting that the composite goals are comprised of the simpler components. Consequently, SLAs contribute to learning goals, and in more complex cases of nested goals, learning goals may ultimately contribute to even more complex goals. The edges bear weights[1], defined by the pedagogical expert who creates the LG, which denote the contribution of a constituent (source) vertex to the target vertex. These structures reside on the MaTHiSiS Learning Content Space (LCS), as detailed in Deliverable **D7.3 *MaTHiSiS platform, 2nd release* [8].**
2. **Personalised LG instances (LGIs)** are created for each learner, as soon as the learner executes a learning experience that involves a particular LG. These instances (graph $G' = (V', E')$) incorporate a reference to their corresponding core unpersonalised counterparts and represent the personal learner models over the parent, core LG. They differ in structure from their respective core graphs in that they allow assignment of a scalar weight to the vertices of the personalised graph, indicating the uptake of the learner on the different learning objective components. They also include a reference to particular learner that has taken up training on the particular competences of the core LG. These structures reside on the MaTHiSiS User Space (US), as detailed in Deliverable **D7.3 *MaTHiSiS platform, 2nd release* [8].**
3. As in the case of SLAIs (cf. Deliverable D3.2 [4]), a historical record of **runtime LG instances** (denoted as 'LGI_rtm') per learner is maintained on the MaTHiSiS cloud database, to enable advanced learning analytics services, as described in Deliverable *D4.5 Multimodal learning analytics* [10]. While (long-term) personal LG instances always reflect the last state of the LGI for a learner, runtime instances are stills in time that capture the state of the LGI at any given key moment of the Learning Experience when the LGI was updated by the personalisation/adaptation processes. Runtime LGIs bear a reference to their long-term LGI counterpart, along with the reference to the particular session that an update took place. These structures also reside on the MaTHiSiS User Space (US), as detailed in Deliverable **D7.3 *MaTHiSiS platform, 2nd release* [8].**

Evidently, LGs' most prominent dependency is the Smart Learning Atoms (SLAs). LG structures encompass a reference to each SLA they contain and LGIs contain references to the corresponding personal Smart Learning Atom Instances (SLAIs, cf. D3.2 [4]), respectively. Progress over particular LGIs advances all constituent competences of the graph (SLAs and goals), and as such it may affect other LGIs of the particular user, if the graphs share common standalone SLAIs. However, if no commonalities exist with other LGs, progress over a particular LGI will not affect any other learning scenario(s).

During the personalisation and adaptation process, all vertex weights of a given LGI are updated through the MaTHiSiS Learning Graph Engine (LGE) [8]. This entails training for the *optimal* atomic learning objective constituents (i.e. SLAs), which in turn implies optimally training for the composite learning goals. This adjustment is based on the learner's personal explicit learning styles, their

[1] weight ∈ [0.0, 1.0]

gradual overall progress over the objectives assumed, the temporal fluctuation of competences. This oscillation is derived by the learners' performance and affective response to the learning activities undertaken. Most importantly, it is also based on the contribution (edge weight) of each learning objective component to influenced learning goals.

Figure 1 graphically illustrates the interdependencies pertaining to LGs, based on the deployment of the MaTHiSiS database schema. The collections that the LGs are related to are portrayed as empty placeholders for visual simplification purposes. This schema remains stable since the first version of this document, apart from the addition of field 'type' in the structure of runtime LGIs.

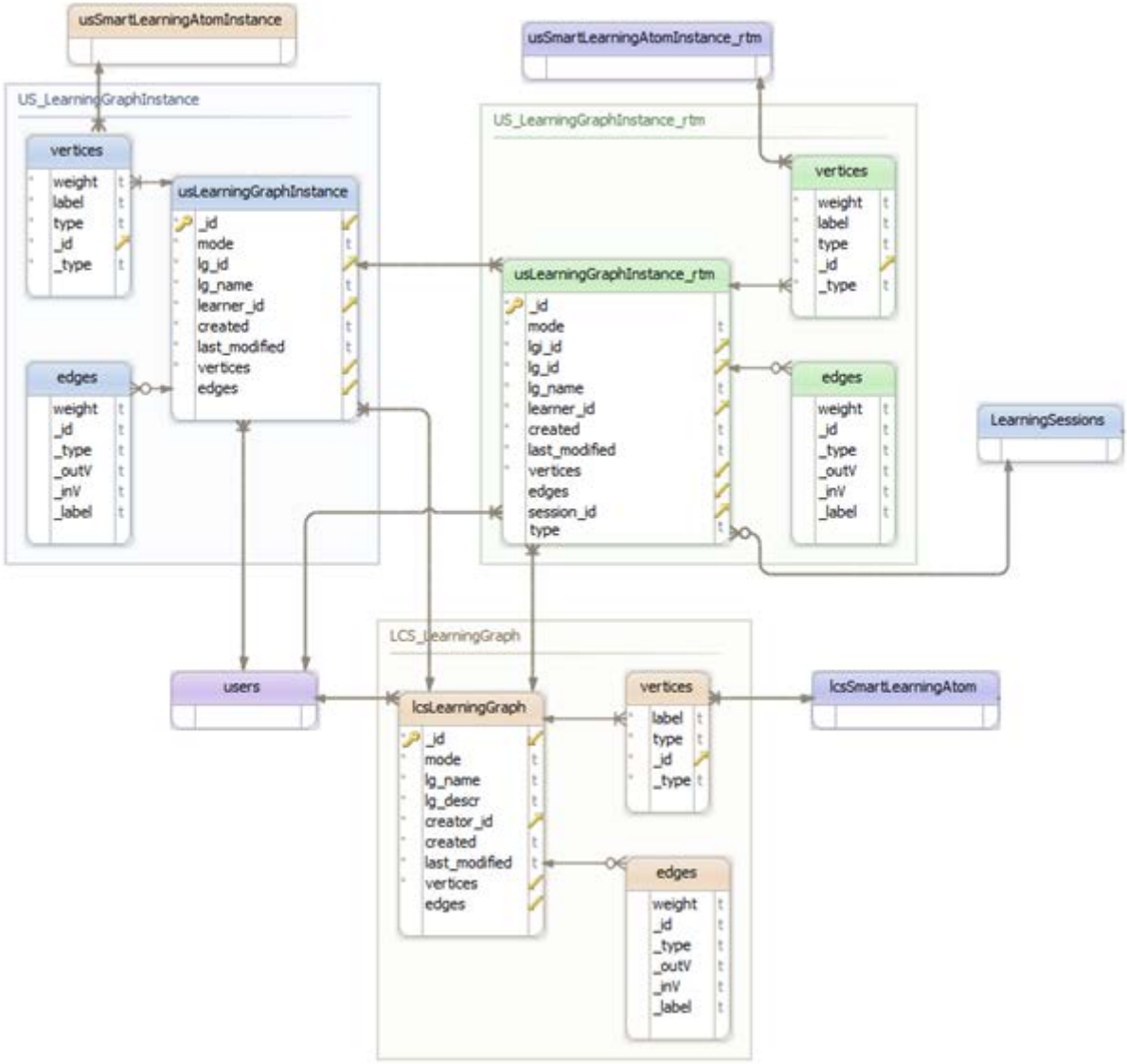

Figure 1: LGs and LG Instances and dependencies with other MaTHiSiS collections

2.3 Learning Graphs educational attributes: concrete examples

The following examples outline the core attributes that pertain to the Smart Learning Atoms, as they were manifested during the MaTHiSiS assisted pilots.

2.3.1 Ubiquity

Ubiquity refers to the capacity of Learning Graphs to holistically represent domain models of skills/knowledge/competences of concrete learning scenarios, and thus enable training of target

learning objectives for all relevant learners under any possible educational setting, context and technological capacities.

During the assisted pilots, 20 Learning Graphs were developed for different thematics or piloting user groups, outlining networks of competences to be trained in MaTHiSiS' Learning Experiences. They pertained to a plurality of interconnected SLAs and learning goals, as portrayed in Figure 2.

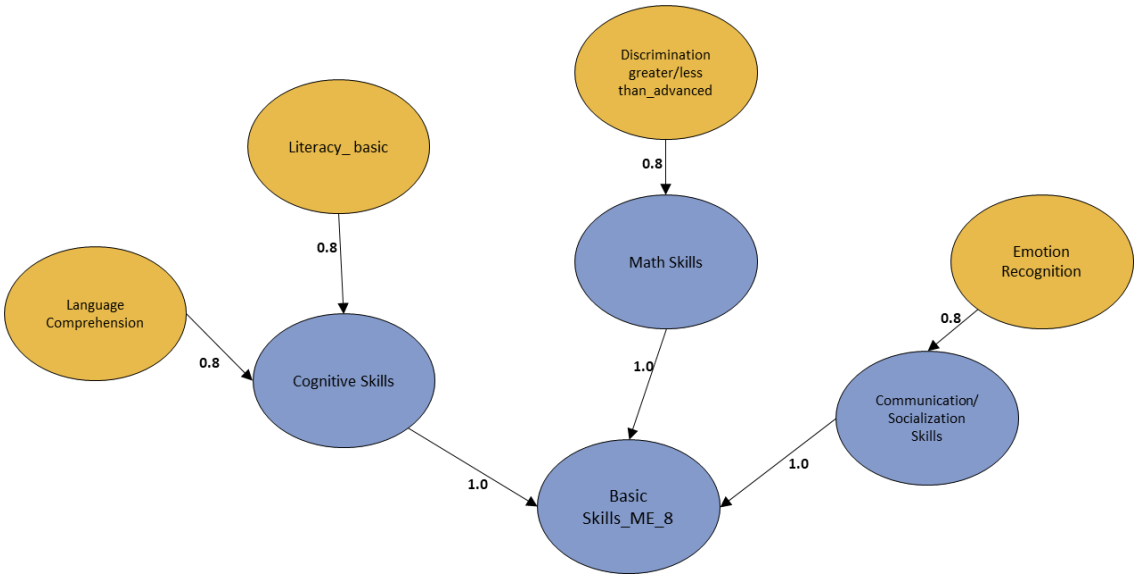

Figure 2: Learning Graph "FMD_ME_8" used by partners FMD in assisted pilots for mainstream learners around 8 years of age

Figure 2 presents a LG where SLAs contribute to different learning goals, while all goals contribute to a central goal, which represents the overall learning objective of the graph and serves as the connective vertex to maintain graph connectivity. This graph served as the foundation to train the skills/knowledge/competences encapsulated in its vertices for several mainstream children aged around 8 years old through a plurality of learning activities, executed in different Platform Agents, such as mobile devices and laptops/desktop PCs, in different settings, depending on the requirements of the learning context.

2.3.2 Individuality

As in the case of SLAs (cf. D3.2), individuality refers to the capacity of the LGs to be instantiated to strictly personal structures (Learning Graph Instances) that carry each learner's personal level of achievement in all pieces of knowledge/skill/competence that the LG represents.

As an example for the MaTHiSiS assisted pilots, the LG depicted in Figure 2, was instantiated in more than 40 personal LGIs for different learners of the platform. In total, over 300 LGIs were created for all learners that participated in the assisted pilots.

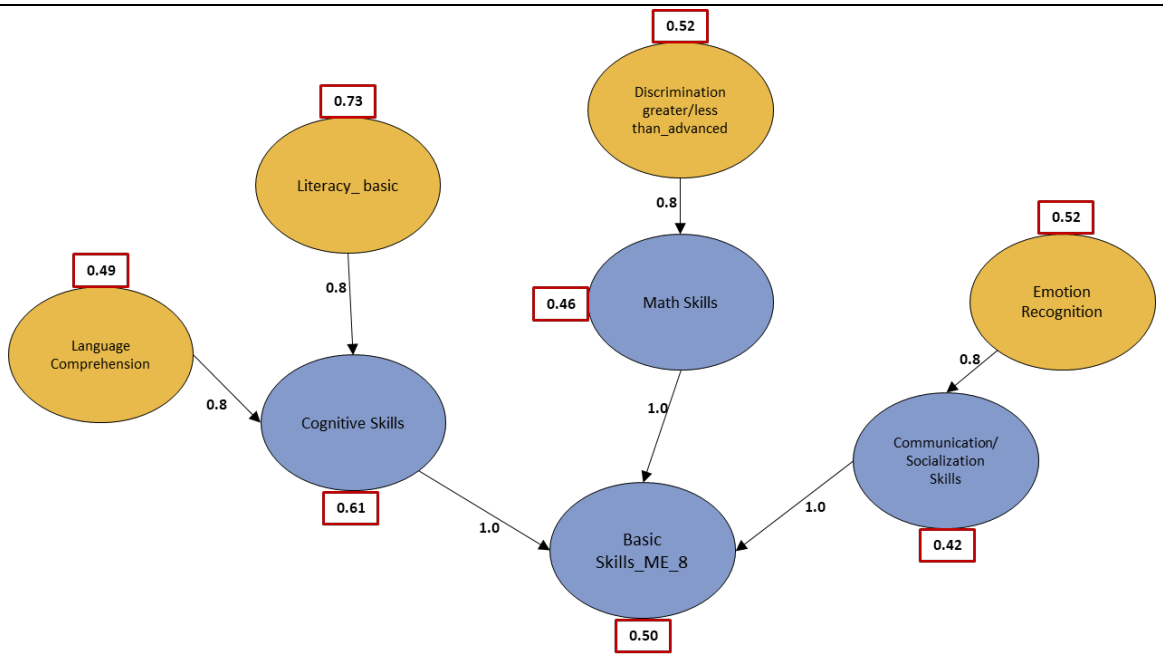

Figure 3: Learning Graph Instance of LG "FMD_ME_8" for a particular learner

2.3.3 Non-linearity

Non-linearity pertains to the ability of the learning objective components of an LG to be trained non-sequentially, based solely on the learner-specific dependencies during a learning experience. In principle, during the MaTHiSiS learning experience, specific learning activities are deployed for a selected SLA, thus training the particular SLA's skill through the designated activity which consequently trains the learner over the more composite learning goals. As detailed in Deliverable D3.6 [6], this selection depends on the SLAIs' achievement level, opting to train the learner first for the less developed SLAs (lowest vertex weight), gradually onto the more developed ones.

Therefore, in the example of Figure 3, the optimal choice for the next iteration of the learning process would be SLA 'Literacy_basic', which bears the lowest weight. After training in this SLA, if the learner's affect state and performance is effective, the corresponding SLAI's weight would rise, thus giving room for training in a completely different skill. Neither the particular SLA nor its contributing learning goal need to be mastered before advancing to another sub-thematic of the LG. In the case of Figure 4, which corresponds to the state of the LGI after a positive learning process iteration, one of the equally weighted SLAs 'Identification of greater/less than_advanced' or 'Emotion recognition' would be pursued next for training.

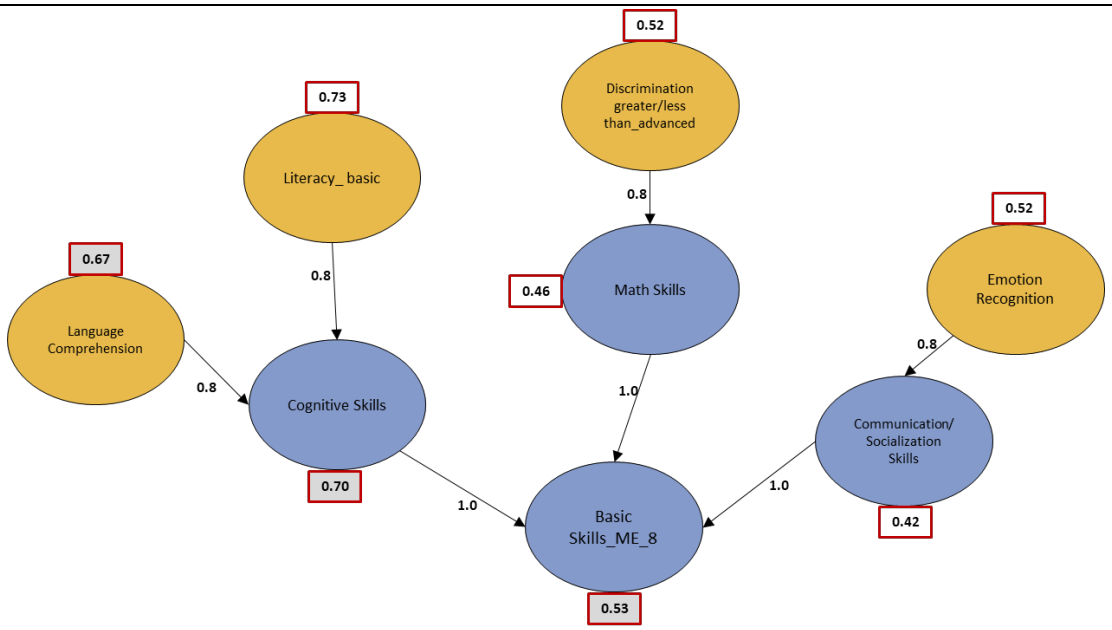

Figure 4: Skill increase of previous LGI

3. Learning Graph library implementation details

3.1 Functionalities

The LG library incorporates all the methods and functionalities required to create, access and modify the LG, LGI and runtime LGI data structures. It is implemented as a Java library which is embedded to the Open API. More specifically, the library offers functionalities to:

- Create an unpersonalised LG, based on an exposed Java method that receives as input the mandatory fields of the data structure (data structure as per Section 6).
- Create an unpersonalised SLA, based on a given JSON input of a serialised SLA (data structure as per Section 6).
- Create a personal LG instance, based on an exposed Java method that receives as input the mandatory fields of the data structure (data structure as per Section 6).
- Create a personal LG instance, based on a given JSON input of a serialised LGI (data structure as per Section 6).
- Create a runtime LG instance, based on a given JSON input of a serialised LGI (data structure as per Section 6), adding the information of the learning session where the LGI was modified in and the method and component responsible for its modification. Methods include creation, reset, personalisation and adaptation. The latter two may be differed to the two components that fulfil these mechanisms, i.e. the DSS and the LGE[2].
- For each non-mandatory field missing from the input (parameters or JSON structure), provide default values to produce a complete data structure.
    - In the case of date/time fields, the current system date and time are set, unless explicitly stated otherwise.
    - In the case of LGI vertex weights, initial default weight (0.3) is set for all vertices in the very first instantiation of a LG to a personal LGI, unless explicitly stated otherwise. This is subject to change in future releases of the personalisation mechanism, where initial weights will be set explicitly based on the constant learner style and capacities, declared explicitly in their learner's profile.
- Retrieve and set (update) different fields of the structures.
    - For all update operations, the 'last modified' field is automatically updated to the current system date and time, unless this field is explicitly declared in the input (parameters or JSON).
- For facilitating the works of the Experience Engine, a direct service to retrieve SLAs of an unpersonalised LG is exposed, without having to explicitly retrieve and parse the entire LG structure.
- Similarly, for facilitating the first step of the personalisation and adaptation process (i.e. the update of SLAI weights only, prior to graph-based adaptation), a direct service to retrieve SLAIs of a personalised LGI is exposed, without having to explicitly retrieve and parse the entire LGI structure.
- Communicate with the Learning Graph engine, in order to update an LGI's vertex weights following personalisation or adaptation.
- Reset an LGI's vertex weights (to 0.3), enabling to re-initiate a particular Learning Experience for a learner. This is an assistive functionality that arose during the driver pilots, where a tutor might need to re-start the Learning Experience for a given learner account anew, either

[2] As detailed in Deliverable D6.2, the DSS is responsible for the first adaptation/personalisation step, i.e. adjustment of individual SLAI weights and the LGE is responsible for the second adaptation/personalisation step, i.e., adjustment of all LGI vertex weights based on the spectral analysis of the graph structure.

due to tests that might have needed to be conducted or due to re-evaluation of environmental settings or modification of the core unpersonalised LG.

- Evaluate validity of LG, LGI and runtime LGI structures as per the mandatory fields.
- Create JSON serialisations for each of the supported data structures (LG, LGI, runtime LGI) to be inserted to the MaTHiSiS DB through the LG lib Open API.

3.2 Open API

The *JAX-RS* [13] Java API for RESTful Web Services was used to create web services according to the Representational State Transfer (REST) architectural pattern for the LG lib Open API. The API is also responsible for serialising, retrieving and deleting LG, LGI and runtime LGI entries to and from the MaTHiSiS database. While the LG lib is in charge of processing (access, creation, update) LG and LGI structures, the Open API is responsible of receiving and transmitting the data to the callers that wish to access the structures in the DB and the libraries' functionalities.

Access to the LG lib Open API is available through the central *<MaTHiSiS base URL>/api/lg/* base URL. MaTHiSiS components that consume LG lib functionalities through the Open API are able to get data from appropriate HTTP connections (bound to specific URLs). The details of the functionalities behind the REST calls available through the LG lib Open API, are listed below and are documented in detail in Appendix II: LG lib Open API documentation.

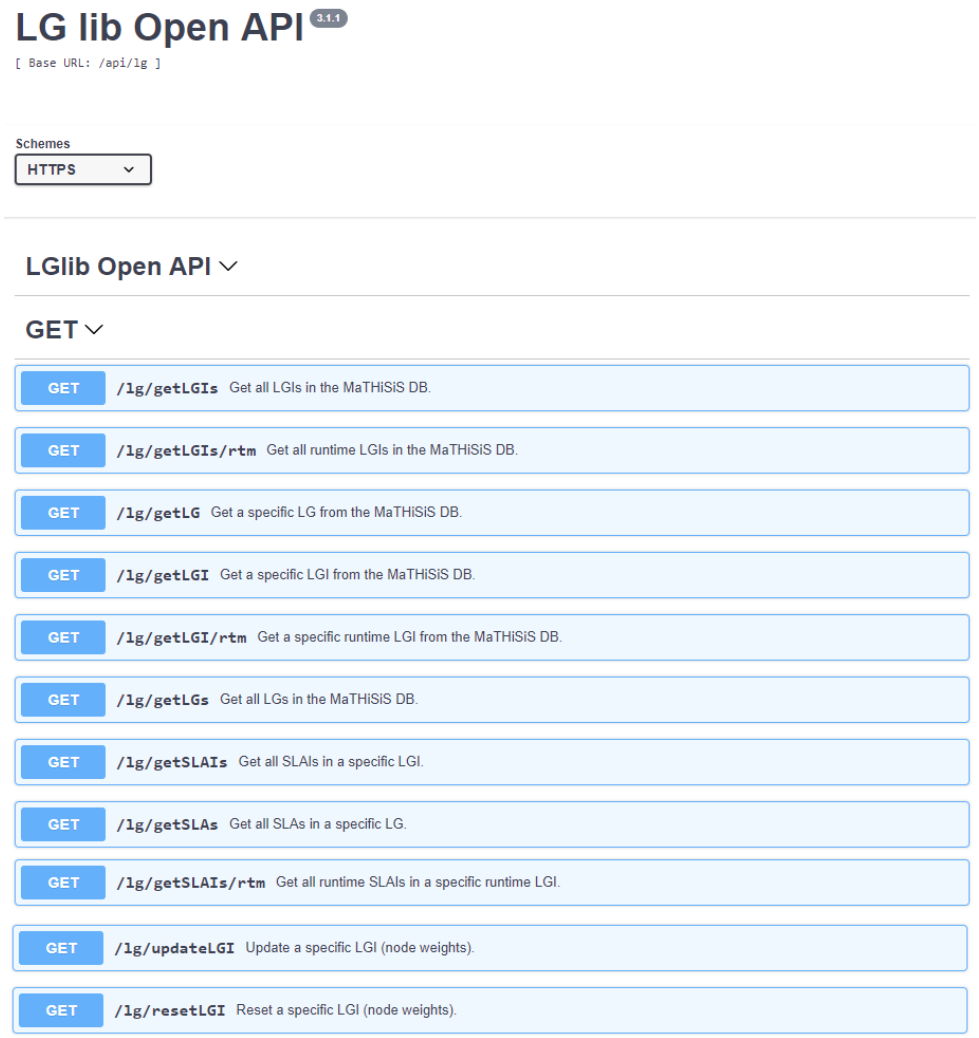

Figure 5: The base URL and GET methods available through the LG lib Open API

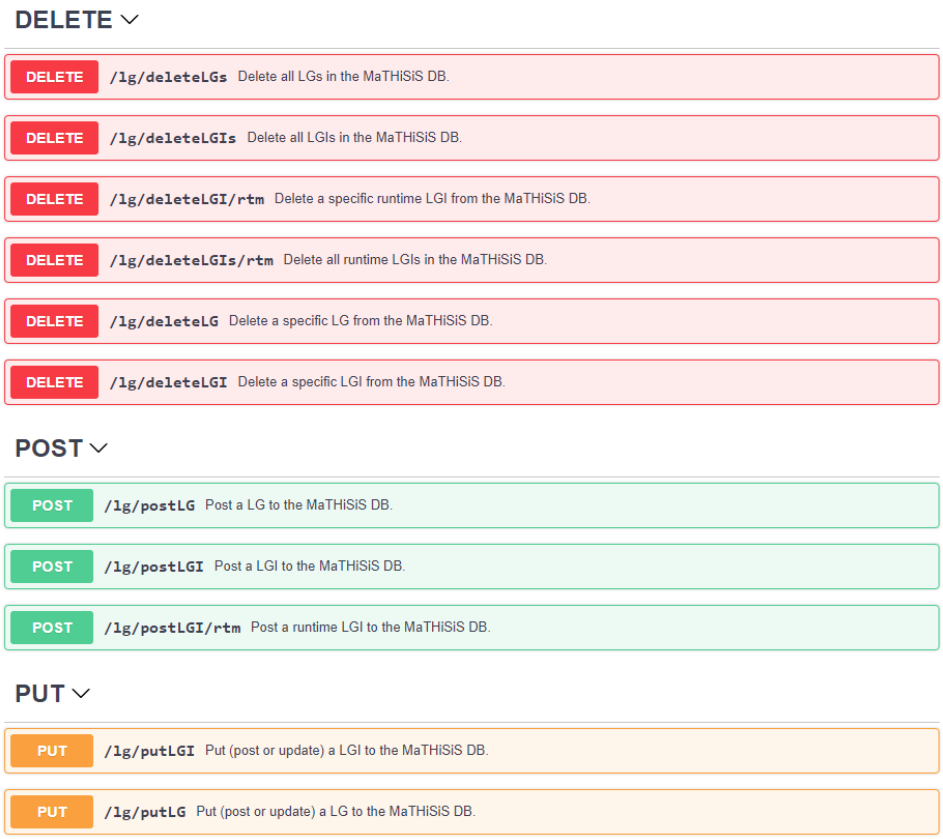

Figure 6: The DELETE, POST and PUT methods available through the LG lib Open API

In short, the LG lib Open API exposes services to:

- Retrieve all LGs, LGIs or runtime LGIs from the MaTHiSiS database. For LGIs and runtime LGIs, filters to retrieve instances specific to a particular core unpersonalised LG or a particular user are available. For runtime LGIs, a filter to retrieve runtime LGIs pertaining to a particular learning sessions is also available.
- Retrieve a specific LG, LGI or runtime LGI from the MaTHiSiS database.
- Retrieve all SLAs, SLAIs or runtime SLAIs attached to a particular LG, LGI or runtime LGI respectively. This is an assistive functionality for back-end components (namely the Experience Engine, the DSS and the Learning Analytics Dashboard respectively), so that they can access the SLAs/SLAIs/runtime SLAIs without having to draw up and parse the entire LG/LGI/runtime LGI structures.
- Delete all LGs, LGIs or runtime LGIs from the MaTHiSiS database. For LGIs and runtime LGIs, filters to delete instances specific to a particular core unpersonalised LG or a particular user are available. For runtime LGIs, a filter to delete runtime LGIs pertaining to a particular learning sessions is also available.
- Delete a specific LG, LGI or runtime LGI from the MaTHiSiS database.
- Post (insert) a LG, LGI or runtime LGI to the MaTHiSiS database. These functionalities check if the particular structure to be posted already exists in the DB and reject the insertion if so. They also implement structure validation to ensure that the structure to be inserted complies with the defined LG, LGI, runtime LGI structure models, per case. For LGIs, this functionality also evokes the insertion of a runtime LGI entry in DB each time it is called (creation status).
- Put (insert or update) a LG, LGI or runtime LGI to the MaTHiSiS database. These functionalities implement structure validation, to ensure that the structure to be inserted/updated complies with the defined LG, LGI, runtime LGI structure models, per case.
- Update an LGI. This functionality triggers the LGE to conduct graph-based adaptation of an LGI's vertex weights [8]. It is called by the DSS after the first step of the personalisation/adaptation is

concluded and interfaces communication between the LGE and the LG lib in order to update vertex weights of a given LGI after the LGE processing has concluded. This functionality also evokes the insertion of two runtime LGI entries in DB each time it is called, one on initialization to denote that the DSS update has concluded and one after the LGE update has concluded.

- Reset an LGI. This is an assistive call, called through explicit use of the Learning Experience Supervisor's [7] 'Reset' button. It triggers the respective reset functionality of the LG lib and evokes the insertion of a runtime LGI entry in DB each time it is called (reset status).

3.3 Interface with the Front-end

Learning Graphs are the key concept of the pedagogical methodology introduced in MaTHiSiS and play a central role in the definition of the goals and activities of a Learning Experience. They intervene in several places of the MaTHiSiS Front-end:

- In the Learning Content Manager (LCE), all the LGs created by MaTHiSiS users, stored in the Learning Graphs Repository (LGR), can be browsed, viewed and edited.
- The LCE, where tutors can create and publish new LGs, as well as edit and update existing ones.
- In the Learning Experience Supervisor, LGs can be seen by tutors and learners in their Learning Experiences.

3.3.1 LGs in the Learning Content Editor

The main functionality of the LCE is to give tutors the tools to create and edit MaTHiSiS-related content. The LCE development has been focused on this core functionality. LGs management through LCE can be broken down in three main objectives:

- Provide a tool for creating and editing LGs, with Learning Goals and Smart Learning Atoms (SLAs);
- Ensure the compatibility of these tools with the defined LG data model;
- Establish the communication (read/write) in the LCE using the LG lib Open API for LGs.

Multiple users can take part during the content creation process in MaTHiSiS, working on the different building blocks (LG, SLA, LA, LAM and LM). Manipulate all these concepts in a single, unified edition tool would potentially lead to a difficult and not easy to use tool. An early decision in the project has been taken to create separate and independent tools for each, with the remaining needs of interconnection between these edition tools. Furthermore, a quick navigation between them seems mandatory, making the work on different elements during the same content creation process more pleasant.

Based on this reflexion, two additional goals are thus added for the LCE:

- Edition tools must be simple and focused on a single concept;
- Navigation between the different edition tools must be quick and easy, but not required at all times to create content.

A holistic overview of the entire Learning Content Editor is provided in the Deliverable *D3.9 MaTHiSiS Frontend Components* [7].

In early period of the project, UI mock-ups for the different LCE tools were created to reach these goals, in order to further refine the list of functionalities required. They were shared with the consortium, who provided their feedback in terms of appearance, usability and functionalities. The approach taken for UI design has been to work on mock-ups for the entire tool set (LG Editor, SLA Editor, LA Editor, LAM Editor, LM Editor) at the same time to ensure that they all have the same look and feel and that they are inline in terms of functionalities implemented in the different tools.

3.3.1.1 LG Editor

The LG editor has evolved over the time; some screenshots below illustrate the different steps the editor has been through. Figure 7 presents a screenshot of the UI mock-up for the LG Editor tool designed during the 1st year:

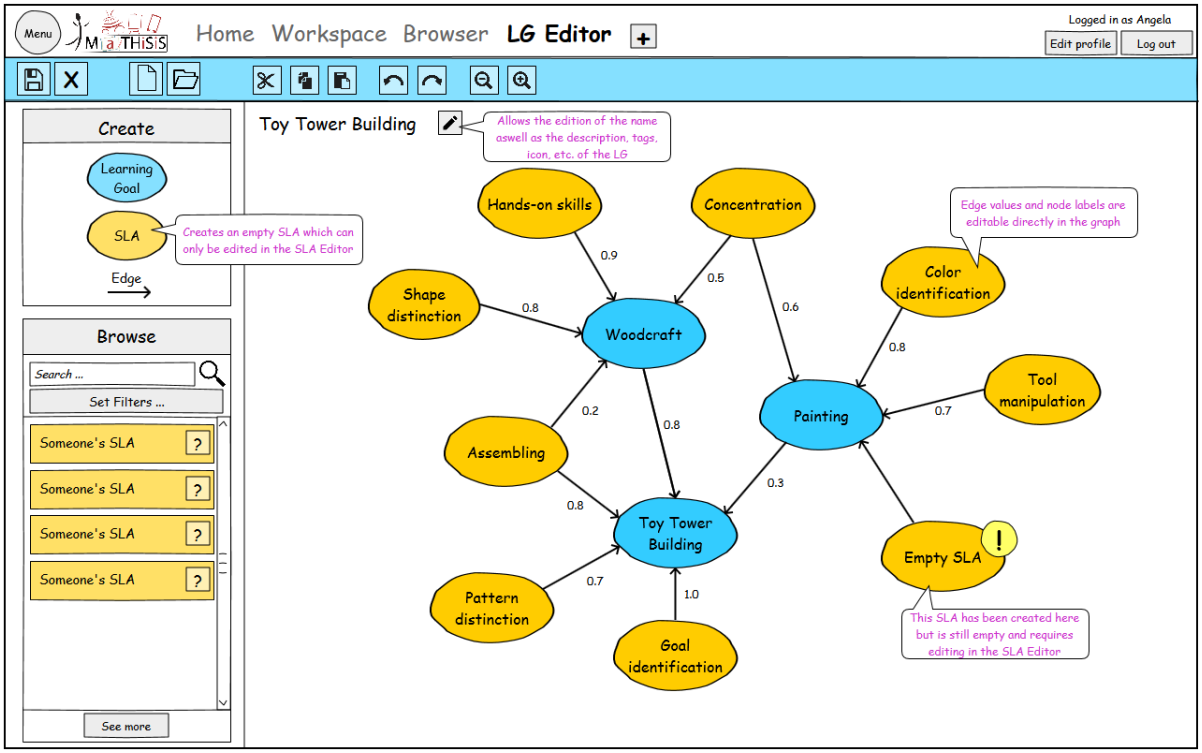

Figure 7: UI mock-up for the LG Editor

Figure 8 presents the first prototype of the LG Editor developed during the 1st year :

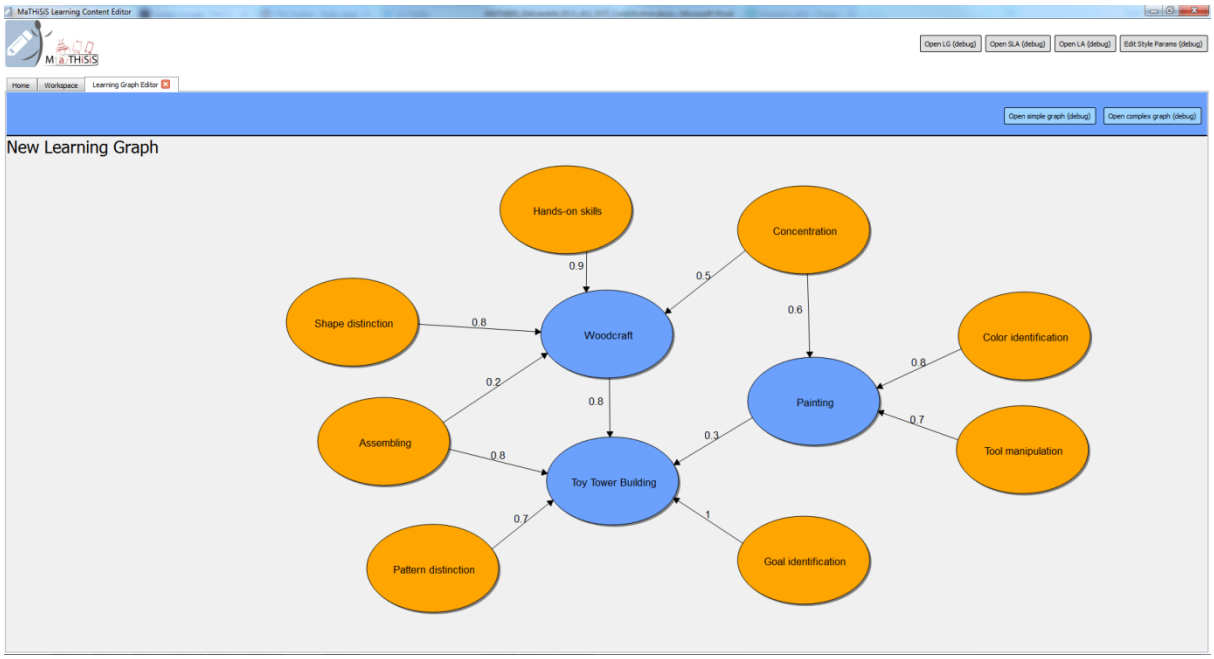

Figure 8: First prototype of the LG Editor

Figure 9 shows the current status of the LG Editor after the 2nd year:

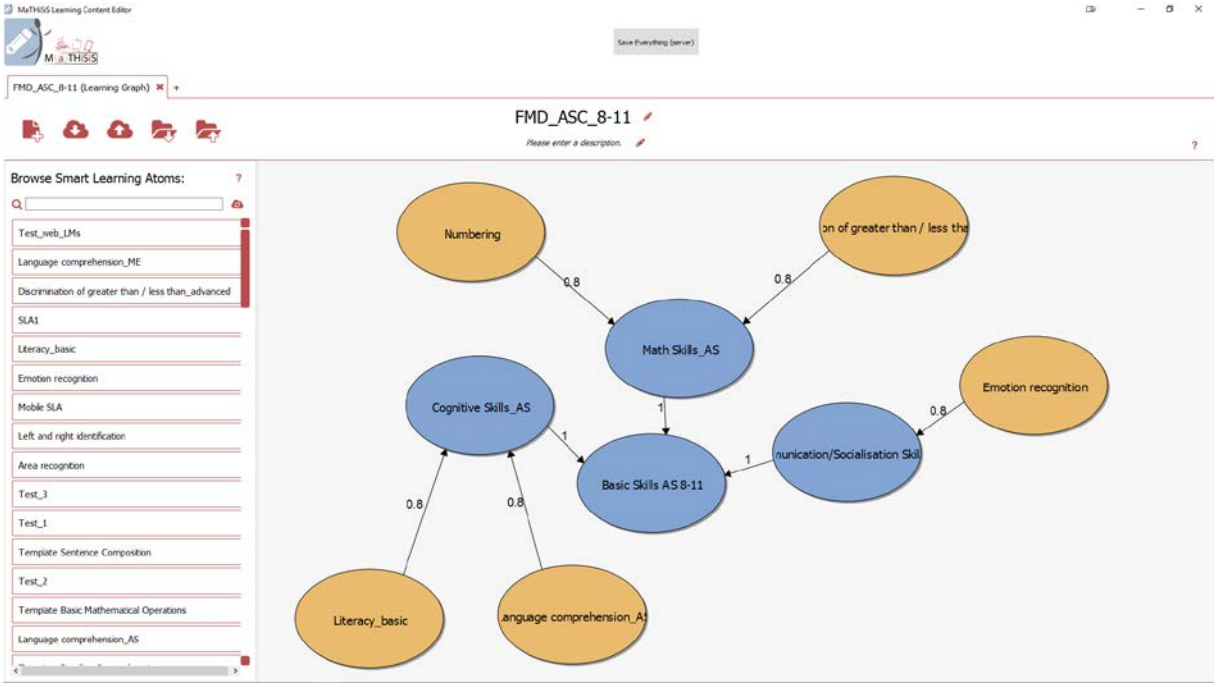

Figure 9 : Current state of LG Editor

Learning Graph Editor displays the graphs with two different types of nodes:

- Learning Goals (here, in blue);
- Smart Learning Atoms (SLAs, here in orange).

Connections between the nodes that are the Learning Goals and SLAs are made with oriented, weighted edges, the weight representing the importance of a SLA or a Learning Goal towards another Learning Goal. As aforementioned, edges starting from a Learning Goal towards a Smart Learning Atoms should and would not be created, as nothing contributes to SLAs. Edges can be drawn between two existing nodes, and their weight edited directly in the graph.

The user can quickly create new empty Learning Goals and SLAs with the Create functionality available through a simple right click in the working zone. Learning Goals do not have a specific editor attached, as they are simply a label, so they can be edited at will in the graph. Simple edition for the SLAs is available through the LG Editor, which is mostly renaming. For an extensive editing SLAs must be opened up in a SLA Editor in a new tab. An empty SLA will need further edition before it can be published on the MaTHiSiS cloud, but letting the user completely define his LG in just one time without having to constantly swap to the SLA Editor was an important point that mattered in this design decision. Once the LG is completed, even with empty SLAs, the user can publish their LG to the cloud with the "Save to the cloud" option available in the toolbar.

Publishing a LG to the MaTHiSiS cloud is done using the LG Open API method POST at the address */api/lg/postLG*. The implementation of this method is described in the OpenAPI specification and is presented in this document at section 3.2

The Browse panel on the left allows the user to view and access the SLAs stored on the MaTHiSiS cloud, and to add existing ones (made by them or by other MaTHiSiS users) to his/her LG, exploiting the reusability concept.

SLAs are retrieved in the Browse panel using the SLA Open API method GET at the address */api/sla/getSLAs*. This method is described in detail in the ***D3.2 - MaTHiSiS Smart Learning Atoms 24.***

Apart from Saving to and Downloading from the Cloud, the toolbar offers an option to save the draft LG (under edition) to the disk as local copy, which can be useful in the case the user loses the connection with the MaTHiSiS cloud during a content creation session. In the same way, it is possible to open an existing SLA from a local copy.

3.3.2 LGs in the Learning Experience Supervisor

The Learning Experience Supervisor (LES) offers the possibility to tutors to manage ongoing Learning Sessions for each of their learners. LES also gives the possibility to both tutors and learners to review their entire Learning Experiences, down to every single Learning Session. The former functionality is available through the Learning Experience Controller module, while the latter is available through the Analytic Dashboard module. These two modules are described in the next subsection.

3.3.2.1 Learning Analytic Dashboard

A Learning Experience is mainly defined by a Learning Graph, defining its main goals and how to achieve them. Personalized for each learner, the LGs become an excellent way of tracking the learners' progress throughout the entire experience, both for the learner, the tutor and the MaTHiSiS decision support system.

Whenever the user interacts with MaTHiSiS platform during a Learning Experience, LGs must be clearly accessible, as they have a central position, and so have the attached SLAs. Figure 10 shows how the Analytic Dashboard displays information about ongoing or finished Learning Experience. It provides a good example of what information can be seen for the user for each experience.

Detailed information about the Analytic Dashboard can be found in the Deliverable *D4.5 - Multimodal learning analytics* [1].

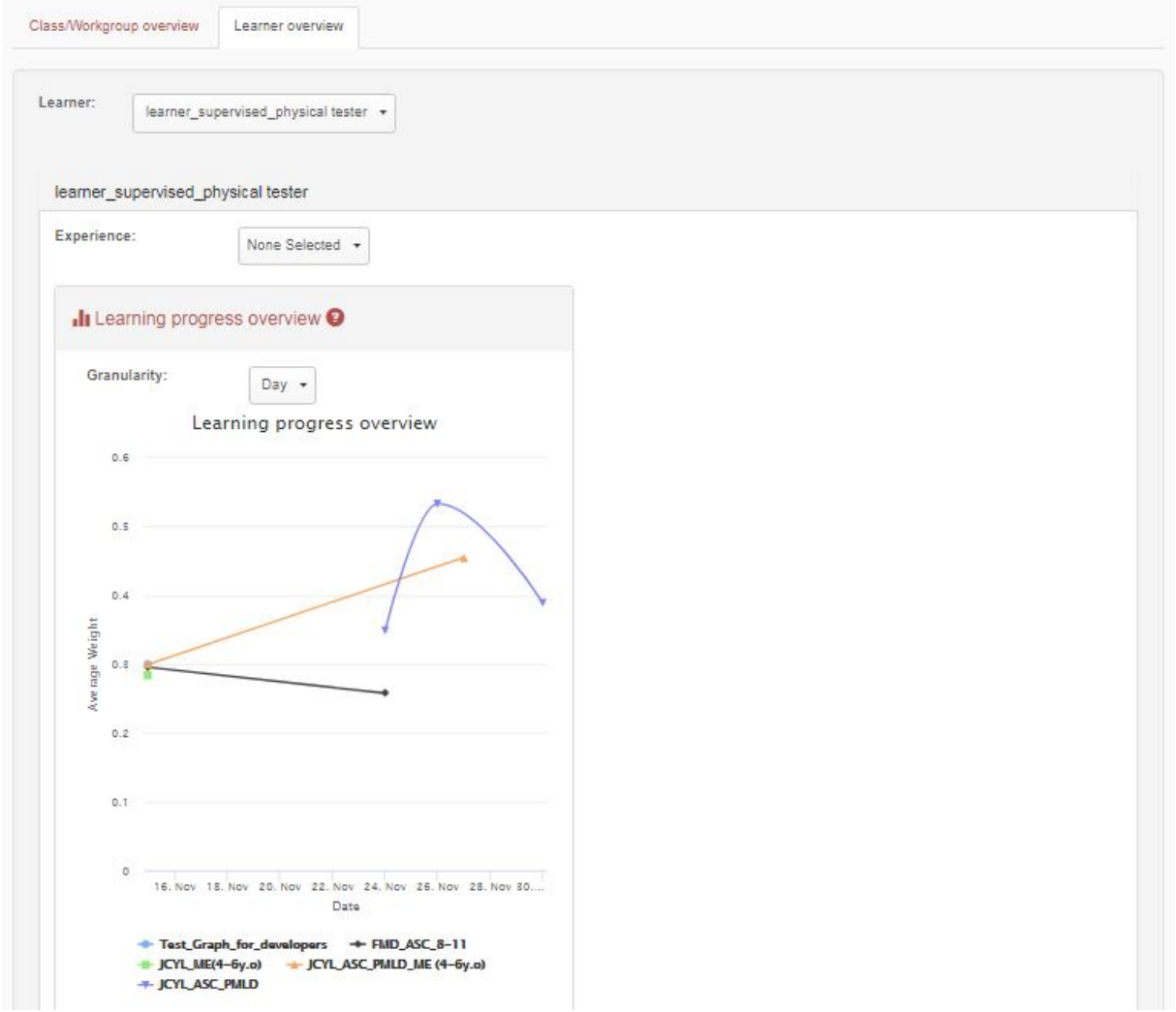

Figure 10: Learning Analytic Dashboard

3.3.2.2 Learning Experience Controller

Figure 11 shows the Learning Experience Controller, part of the Learning Experience Supervisor.

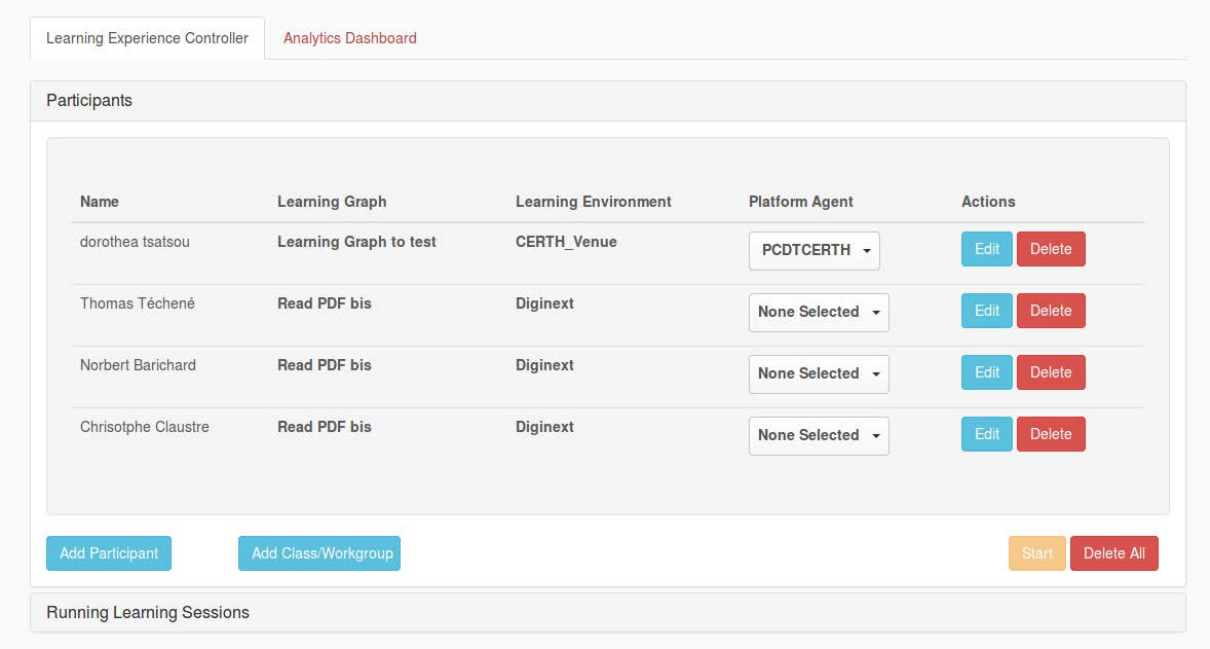

Figure 11: Learning Experience Controller

A list of learners is displayed in the Learning Experience Controller, where a tutor is able to interact with the different Learning Sessions of each learner. For a Learning Session to start, the tutor must choose:

- The location (Learning Environment) where the Learning Sessions will take place.
- A single Platform Agent for each learner. The Learning Materials pertaining to the Learning Experience that takes place will be materialized on this Platform Agent. Note that a Platform Agent can only run one Learning Session at the time.
- A Learning Graph which represents the competences to be acquired during the Learning Experience. The same Learning Graph can be used by several learners at the same time, since personal LGIs are dedicated to each learner onwards.

The tutor may add to this list as many learners as desirable (e.g. add a full class of students). There exist two possibilities: either the tutor adds learners one by one, making a single configuration per learner, or he/she may add a Class/Workgroup, which represents the population of e.g., a given class.

The tutor is also able to *edit* and *delete* each of the displayed Learning Sessions.

The *Start* button placed on the bottom-left corner acts as the trigger to launch the Learning Experiences of the Learners. Moreover, the *Delete all* button, clears all the Learning Sessions pending to be started.

Figure 12 shows the UI for running Learning Sessions. The tutor is able to pause/resume/stop/reset the Learning Session of a learner. The View status functionality offers extra information for the running Learning Session such as the Current Learning Action or Current Learning Material, as well as the current state of the personal LGI in a graphical representation (Figure 13). More information about the Learning Experience Supervisor may be found in *D3.9 - MaTHiSiS Frontend Components* [7].

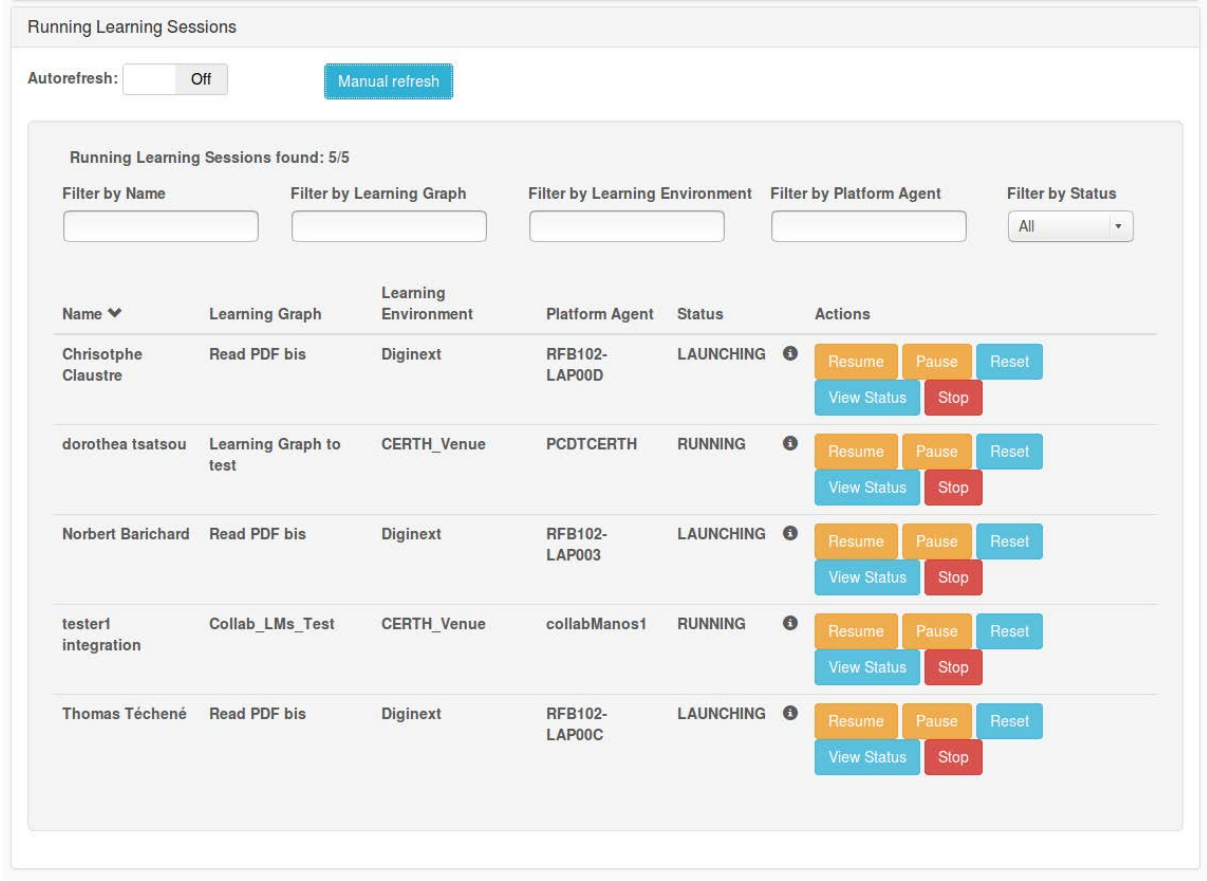

Figure 12: Running Learning Sessions

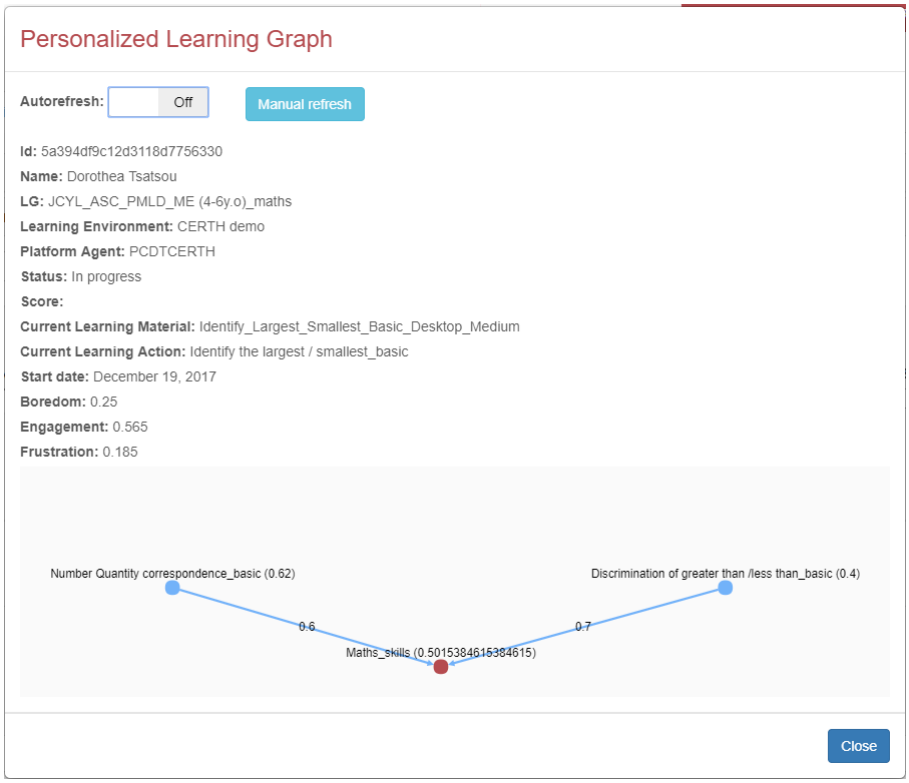

Figure 13: Learning Session status and LGI progression illustration

4. Conclusion

This document detailed the novel educational objective modelling structure introduced in MaTHiSiS, namely the MaTHiSiS Learning Graphs (LGs) and final operations of *Task 3.2 - Learning Graphs Implementation*. This structure is able to represent both the domain model of learning objectives as well as the personal student/trainee models for non-linear, omnipresent learning scenarios.

As anticipated since the first iteration of this document, the conceptual backbone, approach and architecture of the LG-related structures has remained the same, therefore this document's iteration has provided a summary of the main concepts, further grounding them upon concrete use cases as per the MaTHiSiS assisted pilots. Furthermore, it has presented enhancements in the implementation of the structures and of the tools developed for creating, accessing and handling the LG models, based on experiences drawn from the MaTHiSiS driver and assisted pilots.

Until the conclusion of Task 3.2, some improvements of the LG-related tools are provisioned, including usability enhancements of the graphical interfaces related to Learning Graphs, as well as few non-crucial technical enhancements of the LG lib and its Open API, as per the final feedback derived from the assisted pilots, in order to optimise the task's provided tools and functionalities for the real-life pilots.

5. References

[1] Nottingham Trent University (ed.): D4.5 *Multimodal learning analytics*. Deliverable of the MaTHiSiS project, 2017.

[2] DIGINEXT (ed.): D2.4 *Full system architecture*. Deliverable of the MATHISIS project, 2017.

[3] Centre For Research and Technology Hellas (ed.): D3.3 *The MaTHiSiS Learning Graphs M12*. Deliverable of the MaTHiSiS project, 2017.

[4] DIGINEXT (ed.): D3.2 *The MaTHiSiS Smart Learning Atoms M24*. Deliverable of the MaTHiSiS project, 2017.

[5] Centre For Research and Technology Hellas (ed.): D3.3 *The MaTHiSiS Learning Graphs M12*. Deliverable of the MaTHiSiS project, 2016.

[6] DIGINEXT (ed.): D3.6 *Experience Engine M24*. Deliverable of the MaTHiSiS project, 2017.

[7] DIGINEXT (ed.): D3.9 MaTHiSiS Frontend Components. Deliverable of the MaTHiSiS project, 2017.

[8] University of Maastricht (ed.): D6.2 *The MaTHiSiS Learning Graph Engine*. Deliverable of the MaTHiSiS project, 2017.ATOS (ed.): D7.3 *MaTHiSiS platform, 2nd release*. Deliverable of the MATHISIS project, 2017.

[9] MaTHiSiS Description of Action, 2017.

[10] Nottingham Trent University (ed.): D4.5 Multimodal learning analytics. Deliverable of the MaTHiSiS project, 2017.

[11] The Glossary of education reform: Learning Experience, http://edglossary.org/learning-experience/. Last accessed on 21/12/2017.

[12] The TEL Thesaurus and Dictionary meta-project: Learning scenario, http://www.tel-thesaurus.net/wiki/index.php/Learning_scenario. Last accessed on 21/12/2017.

[13] JAX-RS specification: https://jax-rs-spec.java.net/. Last accessed on 21/12/2017.

6. Appendix I: LG Data Structures

The structures of the unpersonalised LGs, personal LG instances and runtime LG instances have remained mostly stable since the first iteration of this document, with minor updates pertaining to values' types, and are detailed in the tables below. An added field has also been added to runtime LGIs, namely 'type', denoting the process that has evoked the creation of a runtime LGI. The data structures are also presented, in context with the rest of the platform's structures, in Deliverable D7.3 [8]. As explained in Section 2, the unpersonalised LGs reside on the Learning Content Space (LCS) and the personalised instances (LGIs and runtime LGIs) on the User Space (US).

LGs, LGIs and runtime LGIs must comprise of at least two vertices, an SLA/SLAI/runtime SLAI (respectively) and a learning goal, as well as at least one connecting edge between two vertices. LGs/LGIs/runtime LGIs need to pertain of connected graphs, i.e. there is a path from any point to any other point in the graph. Serialisation of LGs, LGIs and runtime LGIs follows the GraphSON[3] format, which is a standard JSON-based format for representing graphs and attributes of their comprising elements (i.e. vertices and edges).

All data fields (i.e. column "Key") are mandatory to the structures, however fields marked with (*) should be obligatorily declared upon creation or update. All other fields may be explicitly declared or automatically populated with default/initial/current/empty values by the LG lib. All structures' '_id's are generated automatically upon creation of a new LG/LGI/runtime LGI entry on the MaTHiSiS Mongo DB.

lcsLearningGraph			
Key	Description	Value	Related to
_id	The unique identifier of the LG	ObjectId	-
mode	GraphSON format required field. Always set to "NORMAL".	String Default*: "NORMAL"*	-
lg_name*	The label of the LG	String	-
lg_descr	Details about what this LG is about	String	-
creator_id*	The unique identifier of the user (tutor) that created this LG	ObjectId	Collection *users*: **_id** Only of *role: Tutor*
created	The date and time that this LG was first created	ISODate	-
last_ modified	The date and time that this LG was last modified	ISODate	-
vertices*	The list of vertices that take part in the particular LG. Vertices encapsulate their own set of attributes, listed hereafter as vertices.*X.*	Array	-
vertices.label*	The unique name of the vertex.	String	Collection

[3] https://github.com/tinkerpop/blueprints/wiki/GraphSON-Reader-and-Writer-Library

	If the vertex represents an SLA, then this name must be the same as the corresponding unique SLA_NAME.		*lcsSmartLearningAtom:* **SLA_NAME**
vertices.type*	The type of the vertex in terms of type of learning content component. Can be only one of "*SLA*" or "*LEARNING_GOAL*"	String Options: *"SLA"* or "*LEARNING_GOAL"*	
vertices._id*	The unique identifier for the particular vertex. If the vertex represents an SLA, then this name must be the same as the corresponding unique identifier of the SLA's *_id* in collection *lcsSmartLearningAtom*. If it represents a learning goal, then a random hex identifier is assigned to it.	ObjectId* *or* String (hex)** *For *vertices.type = SLA* **For *vertices.type = LEARNING_GOAL*	Collection *lcsSmartLearningAtom:* **_id** Only for *vertices.type: SLA*
vertices._type	GraphSON format required field. Always set to "vertex".	String Default*: "vertex"*	-
edges*	The list of edges that take part in the particular LG. Edges encapsulate their own set of attributes, listed hereafter as edges.*X.*	Array	-
edges.weight*	A weight in [0, 1] that denotes the contribution degree of the source vertex to the target vertex.	Double Default: 1.0	-
edges._id	The unique identifier for the particular edge. Edge identifiers are essentially a sequential enumeration of the graphs edges from 0 to n, n being the total number of edges in the graph.	Integer	-
edges._type	GraphSON format required field. Defines the type of the graph component.	String Default*: "edge"*	-

	Always set to “edge”.		
edges._outV*	The unique vertices._id that this edge is directed *from* (i.e. outwards direction). It can be either a vertex of vertices.type “SLA” or of vertices.type “LEARNING_GOAL”.	String (hex)	*(this)*: **vertices._id**
edges._inV*	The unique vertices._id that this edge is directed *to* (i.e. inwards direction). It can ONLY be a vertex of vertices.type “LEARNING_GOAL”.	String (hex)	*(this)*: **vertices._id**
edges._label	GraphSON format required field. The label of the edge. In MaTHiSiS, LG edges are not labelled, therefore this field is always set to “default”.	String Default*: “default”*	-

Table 2: Unpersonalised LG data structure (lcsLearningGraph)

usLearningGraphInstance			
Key	Description	Value	Related to
_id	The unique identifier of the LGI	ObjectId	-
mode	GraphSON format required field. Always set to “NORMAL”.	String Default*: “NORMAL”*	-
lg_id*	The unique identifier of the corresponding unpersonalised LG.	ObjectId	Collection *lcsLearningGraph*: **_id**
lg_name	The label of the of the corresponding unpersonalised LG	String Same as *lcsSmartLearningAtom:* **lg_name**	-
learner_id*	The unique identifier of the user (learner) that this LGI applies to.	ObjectId	Collection *users*: **_id** Only of *role: Learner*
created	The date and time that this LGI	ISODate	-

	was first created		
last_ modified	The date and time that this LGI was last modified.	ISODate	-
vertices*	The list of vertices that take part in the particular LGI. They always correspond to the core LG's vertices. Vertices encapsulate their own set of attributes, listed hereafter as vertices.*X.*	Array	-
vertices.weight*	A weight in [0,1] that denotes the uptake of the learner for this particular vertex. Initial default value is 0.3.	Double Default initial*: 0.3*	-
vertices.label*	The unique name of the vertex. If the vertex is of type SLA, then this name must be the same as the corresponding unique SLA_NAME.	String	Collection *lcsSmartLearningAtom:* **SLA_NAME** = Collection *usSmartLearningAtom:* **SLAI_NAME**
vertices.type*	The type of the vertex in terms of type of learning content component. Can be only one of "*SLA*" or "*LEARNING_GOAL*"	String Options: *"SLA"* or *"LEARNING_GOAL"*	
vertices._id	The unique identifier for the particular vertex. If the vertex represents an SLAI, then this name must be the same as the corresponding unique identifier of the SLAI's *_id* in collection *usSmartLearningAtomInstance*. If it represents a learning goal, then a random hex identifier is assigned to it.	ObjectId* *or* String (hex)** *For *vertices.type = SLA* **For *vertices.type = LEARNING_GOAL*	Collection *usSmartLearningAtomInstance:* **_id** Only for *vertices.type: SLA*
vertices._type	GraphSON format required field. Always set to "vertex".	String Default*: "vertex"*	-
edges*	The list of edges that take part in the particular LGI. They always correspond to the	Array	-

	core LG's edges. Edges encapsulate their own set of attributes, listed hereafter as edges.*X*.		
edges.weight*	A weight in [0, 1] that denotes the contribution degree of the source vertex to the target vertex.	Double	-
edges._id	The unique identifier for the particular edge. Edge identifiers are essentially a sequential enumeration of the graphs edges from 0 to n, n being the total number of edges in the graph.	Integer	-
edges._type	GraphSON format required field. Defines the type of the graph component. Always set to "edge".	String Default*: "edge"*	-
edges._outV*	The unique vertices._id that this edge is directed *from* (i.e. outwards direction). It can be either a vertex of vertices.type "SLA" or of vertices.type "LEARNING_GOAL".	String (hex)	*(this)*: **vertices._id**
edges._inV*	The unique vertices._id that this edge is directed *to* (i.e. inwards direction). It can ONLY be a vertex of vertices.type "LEARNING_GOAL".	String (hex)	*(this)*: **vertices._id**
edges._label	GraphSON format required field. The label of the edge. In MaTHiSiS, LG edges are not labelled, therefore this field is always set to "default".	String Default*: "default"*	-

Table 3: Personalised LGI data structure (usLearningGraphInstance)

usLearningGraphInstance_rtm			
Key	Description	Value	Related to
_id*	The unique DB identifier of the LG runtime instance.	ObjectId	-
mode*	Internal GraphSON (i.e. standardised JSON for graphs, cf. Section 3.2) required field. Always set to "NORMAL".	String Default*: "NORMAL"*	-
lgi_id*	The unique DB identifier of the (long-term) last state of the corresponding personalised LGI	ObjectId	Collection *usLearningGraphInstance*: **_id**
lg_id*	The unique DB identifier of the corresponding unpersonalised LG.	ObjectId	Collection *lcsLearningGraph*: **_id**
lg_name*	The unique name of the LG.	String Same as *lcsSmartLearningAtom:* **lg_name** and Same as *usSmartLearningAtomInstance:* **lg_name**	-
learner_id*	The unique identifier of the user (learner) that this LGI applies to.	ObjectId	Collection *users*: **_id** Only of *role: Learner*
session_id*	The unique identifier of the learner session that this LGI's vertex weights were modified during runtime of the learning process	ObjectId	Collection *LearningSessions*: **_id**
type	The MaTHiSiS process that updated the respective LGI and is thus responsible for the creation of this runtime LGI.	String Options: "CREATION" "RESET" "DSS_PERSONALISATION" "DSS_ADAPTATION" "CREATION"	-

		"RESET" "LGE_PERSONALISATION" "LGE_ADAPTATION"	
created	The date and time that this runtime LGI was first created	ISODate	-
last_ modified	The date and time that this runtime LGI was last modified. Maintained for uniformity, runtime LGIs are never modified, only created once. This is always the same date as n field 'created'.	ISODate	-
vertices*	The list of vertices that take part in the particular runtime LGI. They always correspond to the related LGIs vertices, so in extend also to the core LG's vertices. Vertices encapsulate their own set of attributes, listed hereafter as vertices.*X.*	Array	-
vertices.weight*	A weight in [0,1] that denotes the uptake of the learner for this particular vertex.	Double	-
vertices.label*	The unique name of the vertex. If the vertex is of type SLA, then this name must be the same as the corresponding unique SLA_NAME.	String	Collection *lcsSmartLearningAtom:* **SLA_NAME** = Collection *usSmartLearningAtom:* **SLA_NAME** = *usSmartLearningAtom_rtm:* **SLA_NAME**
vertices.type*	The type of the vertex in terms of type of learning content component. Can be only one of "*SLA*" or "*LEARNING_GOAL*"	String Options: *"SLA"* or "*LEARNING_GOAL"*	
vertices._id*	The unique identifier for the particular vertex. If the vertex represents a runtime SLAI, then this name must be the	ObjectId* *or* String (hex)** *For *vertices.type = SLA*	Collection *usSmartLearningAtomInstance_rtm:* **_id**

	same as the corresponding unique identifier of the runtime SLAI's *_id* in collection *usSmartLearningAtomInstance_rtm*. If it represents a learning goal, then a random hex identifier is assigned to it.	**For *vertices.type = LEARNING_GOAL*	Only for *vertices.type: SLA*
vertices._type*	GraphSON format required field. Always set to "vertex".	String Default*: "vertex"*	-
edges*	The list of edges that take part in the particular runtime LGI. They always correspond to the related LGIs edges, so in extend also to the core LG's edges. Edges encapsulate their own set of attributes, listed hereafter as edges.*X*.	Array	-
edges.weight*	A weight in [0, 1] that denotes the contribution degree of the source vertex to the target vertex.	Double	-
edges._id*	The unique identifier for the particular edge. Edge identifiers are essentially a sequential enumeration of the graphs edges from 0 to n, n being the total number of edges in the graph.	Integer	-
edges._type*	GraphSON format required field. Defines the type of the graph component. Always set to "edge".	String Default*: "edge"*	-
edges._outV*	The unique vertices._id that this edge is directed *from* (i.e. outwards direction). It can be either a vertex of vertices.type "SLA" or of vertices.type "LEARNING_GOAL".	String (hex)	*(this)*: **vertices._id**
edges._inV*	The unique vertices._id that this edge is directed *to* (i.e. inwards direction).	String (hex)	*(this)*: **vertices._id**

	It can ONLY be a vertex of vertices.type “LEARNING_GOAL”.		
edges._label*	GraphSON format required field. The label of the edge. In MaTHiSiS, LG edges are not labelled, therefore this field is always set to “default”.	String Default*: “vertex”*	-

Table 4: Personalised LGI runtime record data structure (usLearningGraphInstance_rtm)

7. Appendix II: LG lib Open API documentation

The tables below detail the functionalities of the REST calls available through the LG lib Open API. Parameters marked with (*) are mandatory. All ids are ObjectIds, an inherent Mongo value type used to identify DB entries and all dates follow Mongo's ISODate value type.

URL pattern	**GET api/lg/getLGs**		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all LGs in the MaTHiSiS repository		
Responses	HTTP 200 status code if successful		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**
	-	-	{ "lgs": [{ "lg_id":"{lg _id}", "lg _name":"{lg _name}", }, ...] }

Table 5: LG Open API - GET api/lg/getLGs

URL pattern	**GET api/lg/getLG**		
Method	GET		
Content type	Application/JSON		
Description	Return an unpersonalised LG for a given id (and optionally, name as cross-reference)		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no LG ID is provided.		
	HTTP 404 status code if no LG exists with this id OR an empty output had yielded.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**

	id* label	?id={lg_id } & label={lg_name}	{ "_id" : "{lg_id}" "mode" : "NORMAL", "lg_name" : "{lg_name}", "lg_descr" : "{lg description}", "creator_id" : "{tutor_id}", "created" : "{date/time}", "last_modified" : "{date/time}", "vertices" : [{ "label" : "{vertex_name}", "type" : "SLA", "_id" : "{vertex_id}"), "_type" : "vertex" }, "label" : "{vertex_name}", "type" : "LEARNING_GOAL", "_id" :"{vertex_id}"), "_type" : "vertex" }, ...], "edges" : [{ "weight" : "{edge+weight}", "_id" : "{enumeration}", "_type" : "edge", "_outV" : "{outward_vertex_id}", "_inV" : "{inward_vertex_id}", "_label" : "default" }, ...] }

Table 6: LG Open API - GET api/lg/getLG

URL pattern	**GET api/lg/getSLAs**		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all vertices of type “SLA” in the given LG, which include pointers to corresponding lcsSmartLearningAtom structures		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no LG ID is provided.		
	HTTP 404 status code if no LG exists with this id OR an empty output had yielded.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**

	id*	?id={lg_id }	{ "lg_id": {lg_id}, "lg_name": {lg_name}, "slas": [{ "sla_id":"{sla_id}", "sla_name":"{sla_name}", }, ...] }

Table 7: LG Open API - GET api/lg/getSLAs

URL pattern	**GET api/lg/getLGIs**		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all personal LG instances in the MaTHiSiS repository. If a learner id is defined, return the list of all LGIs that this learner works on. If a LG id is defined, return the list of all LGIs that are instantiated from this LG.		
Responses	HTTP 200 status code if successful		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**
	uid lgid	?uid={learner_id} &lgid={lg_id}	{ "lgis": [{ "lgi_id": "{lgi_id}", "lg_id": "{lg_id}", "lg_name": "{lg_name}", "learner_id": "{lg_name}", }, ...] }

Table 8: LG Open API - GET api/lg/getLGIs

URL pattern	**GET api/lg/getLGI**
Method	GET
Content type	Application/JSON
Description	Return a personalised LGI for a particular learner, given the LGI id (and optionally, name as cross-reference). This call also cross-checks the current state of usSmartLearningAtomInstance

<table>
<tr><td></td><td colspan="3">structures for all SLAIs in the LG and updates either the vertex weights in the LGI or the weights of the corresponding SLAIs to the most recent state (according to the most recent ‘last_modified’ date/time in each corresponding structure).</td></tr>
<tr><td>Responses</td><td colspan="3">HTTP 200 status code if successful</td></tr>
<tr><td></td><td colspan="3">HTTP 400 status code if no LGI ID is provided.</td></tr>
<tr><td></td><td colspan="3">HTTP 404 status code if no LGI exists with this id OR an empty output had yielded.</td></tr>
<tr><td></td><td colspan="3">HTTP 500 if the JSON structure of the list failed to be constructed</td></tr>
<tr><td>Parameters</td><td>Name</td><td>URL pattern</td><td>Success Response Model</td></tr>
<tr><td></td><td>id*</td><td>?id={lgi_id }</td><td>{
"_id" : "{lgi_id}"
"mode" : "NORMAL",
"lg_id" : "{lg_id}",
"lg_name" : "{lg_name}",
"learner_id" : "{learner_id}",
"created" : "{date/time}",
"last_modified" : "{date/time}",
"vertices" : [
{
"weight" : "{vertex_weight}",
"label" : "{vertex_name}",
"type" : "SLA",
"_id" : "{vertex_id}"),
"_type" : "vertex"
},
{
"weight" : "{vertex_weight}",
"label" : "{vertex_name}",
"type" : "LEARNING_GOAL",
"_id" :"{vertex_id}"),
"_type" : "vertex"
},
….
],
"edges" : [
{
"weight" : "{edge_weight}",
"_id" : "{enumeration}",
"_type" : "edge",
"_outV" :
"{outward_vertex_id}",
"_inV" : "{inward_vertex_id}",
"_label" : "default"
},
….
]
}</td></tr>
</table>

Table 9: LG Open API - GET api/lg/getLGI

URL pattern	GET api/lg/getSLAIs		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all vertices of type “SLA” in the given (last state) LGI, which include pointers to corresponding usSmartLearningAtomInstance structures.		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no LGI ID is provided.		
	HTTP 404 status code if no LGI exists with this id OR an empty output had yielded.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**
	id*	?id={lgi_id }	{ "learner_id": {learner_id}, "lgi_id": {lgi_id}, "lg_id": {lg_id}, "lg_name": {lg_name}, "slais": [{ "slai_id":"{slai_id}", "sla_id":"{sla_id}", "sla_name":"{sla_name}", }, ...] }

Table 10: LG Open API - GET api/lg/getSLAIs

URL pattern	GET api/lg/getLGIs/rtm		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all personal runtime LG instances in the MaTHiSiS repository. If a learner id is defined, return the list of all runtime LGIs that this learner has worked on. If a LGI id is defined, return the list of all runtime LGIs pertaining to this LGI. If a LG id is defined, return the list of all runtime LGIs pertaining to this LG. If a session id is defined, return the list of all runtime LGIs created during that session.		
Responses	HTTP 200 status code if successful		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**

	uid lgid lgiid sid	?uid={learner_id} &lgid={lg_id} &lgiid={lgi_id} &sid={session_id}	{ "session_id": "{session_id}", "learner_id": "{learner_id}", "lgis_rtm": [{ "lgi_rtm_id": "{lgi_rtm_id}", "lgi_id": "{lgi_id}", "lg_id": "{lg_id}", "lg_name": "{lg_name}", }, ...] }

Table 11: LG Open API - GET api/lg/getLGIs/rtm

URL pattern	**GET api/lg/getLGI/rtm**		
Method	GET		
Content type	Application/JSON		
Description	Return a personalised runtime LGI.		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no runtime LGI ID is provided.		
	HTTP 404 status code if no runtime LGI exists with this id OR an empty output had yielded.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**

	id*	?id={lgi_rtm_id}	{ "_id" : "{lgi_rtm_id}" "mode" : "NORMAL", "lgi_id" : "{lgi_id}", "lg_id" : "{lg_id}", "lg_name" : "{lg_name}", "learner_id" : "{learner_id}", "created" : "{date/time}", "type" : "{type options}", "last_modified" : "{date/time}", "session_id" : "{session_id}", "vertices" : [{ "weight" : "{vertex_weight}", "label" : "{vertex_name}", "type" : "SLA", "_id" : "{vertex_id}"), "_type" : "vertex" }, { "weight" : "{vertex_weight}", "label" : "{vertex_name}", "type" : "LEARNING_GOAL", "_id" :"{vertex_id}"), "_type" : "vertex" }, ….], "edges" : [{ "weight" : "{edge_weight}", "_id" : "{enumeration}", "_type" : "edge", "_outV" : "{outward_vertex_id}", "_inV" : "{inward_vertex_id}", "_label" : "default" }, ….] }

Table 12: LG Open API - GET api/lg/getLGI/rtm

URL pattern	**GET api/lg/getSLAIs/rtm**
Method	GET
Content type	Application/JSON
Description	Return the list of all vertices of type “SLA” in the given runtime LGI, which include pointers to corresponding usSmartLearningAtomInstance_rtm structures.

Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no runtime LGI ID is provided.		
	HTTP 404 status code if no runtime LGI exists with this id OR an empty output had yielded.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Success Response Model**
	id*	?id={lgi_id }	{ "learner_id": {learner_id}, "session_id": {session_id}, "lgi_rtm_id": {lgi_rtm_id}, "lgi_id": {lgi_id}, "lg_id": {lg_id}, "lg_name": {lg_name}, "slais": [{ "slai_rtm_id":"{slai_rtm_id}", "slai_id":"{slai_id}", "sla_id":"{sla_id}", "sla_name":"{sla_name}", }, ...] }

Table 13: LG Open API - GET api/lg/getSLAIs/rtm

URL pattern	**GET api/lg/updateLGI**
Method	GET
Content type	Application/JSON
Description	Update the weight of the vertices for a given personalised LG instance. Is called after the first step of personalisation or adaptation by the DSS. Calls LG lib to update instance goals based on the new SLAI weights. Evokes the second step of personalisation or adaptation through the LGE. Responsible to update SLAI weights after LGE is done (through *POST api/sla/ updateSLAIweight*). Calls the Experience Engine after all updates to trigger materialization of the next step in the Learning Session (*POST /api/LS/learningSessions/{session_id}/materialize*). Responsible to post a runtime LGI as soon as LG lib has updated the LGI's instance goals, with fields DSS_PERSONALIZATION or DSS_ADAPTATION (depending on Learning Session status). Responsible to post a runtime LGI after LGE has updated the LGI, with fields LGE_PERSONALIZATION or LGE_ADAPTATION (depending on Learning Session status).

	Also produces error responses shared with */api/lg/getLGI*, */api/lg/putLGI* and */api/lg/postLGI/rtm*		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no LGI ID is provided OR no session id is provided.		
	HTTP 502 status code if the EE materialization call fails		
Parameters	**Name**	**URL pattern**	**Success Response Model**
	id* sid*	?id={lgi_id} &sid={session_id}	{ {lgi_id} }

Table 14: LG Open API - GET api/lg/updateLGI

URL pattern	**GET api/lg/resetLGI**		
Method	GET		
Content type	Application/JSON		
Description	Reset the weights of all the vertices for a given personalised LG instance to 0.3. Changes all vertex weights to 0.3. Updates corresponding SLAI weights (through *POST api/sla/ updateSLAIweight*). Responsible to post a runtime LGI after update, with field RESET. Also produces error responses shared with */api/lg/getLGI*, */api/lg/putLGI* and */api/lg/postLGI/rtm*		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no LGI ID is provided OR no session id is provided.		
Parameters	**Name**	**URL pattern**	**Input Model**
	id* sid*	?id={lgi_id} &sid={session_id}	{ {lgi_id} }

Table 15: LG Open API - GET api/lg/resetLGI

URL pattern	**POST api/lg/postLG**
Method	POST
Content type	Application/JSON
Description	Create an unpersonalised LG on the MaTHiSiS DB under the lcsLearningGraph

<table>
<tr><td></td><td colspan="3">collection. Missing non-mandatory fields of the input model are automatically filled in with default/initial values.
It also creates corresponding lcsSmartLearningAtom structures in the DB, if not already existent</td></tr>
<tr><td>Responses</td><td colspan="3">HTTP 200 status code if successful</td></tr>
<tr><td></td><td colspan="3">HTTP 400 status code if the input JSON structure is wrong OR no input was provided OR the input structure was not correct OR the LG already exists.</td></tr>
<tr><td></td><td colspan="3">HTTP 404 status code if a creation error failed to insert the structure in the DB.</td></tr>
<tr><td></td><td colspan="3">HTTP 500 if the JSON structure of the list failed to be constructed</td></tr>
<tr><td>Parameters</td><td>Name</td><td>URL pattern</td><td>Input Model
(*)Marks mandatory fields for the input to be accepted</td></tr>
<tr><td></td><td>-</td><td>-</td><td>{
"_id" : "{lg_id}"
"mode" : "NORMAL",
*"lg_name" : "{lg_name}",
"lg_descr" : "{lg description}",
*"creator_id" : "{tutor_id}",
"created" : "{date/time}",
"last_modified" : "{date/time}",
*"vertices" : [
{
* "label" : "{vertex_name}",
*"type" : "SLA",
*"_id" : "{vertex_id}"),
"_type" : "vertex"
},
*"label" : "{vertex_name}",
*"type" : "LEARNING_GOAL",
"_id" :"{vertex_id}"),
"_type" : "vertex"
},
....
],
*"edges" : [
{
*"weight" : "{edge+weight}",
"_id" : "{enumeration}",
"_type" : "edge",
*"_outV" :
"{outward_vertex_id}",
*"_inV" : "{inward_vertex_id}",
"_label" : "default"
},
....
]
}</td></tr>
</table>

Table 16: LG Open API - GET api/lg/postLG

URL pattern	**POST api/lg/postLGI**		
Method	POST		
Content type	Application/JSON		
Description	Create or update a personalised LG instance for a given learner on the MaTHiSiS DB under the usLearningGraphInstance collection. Missing non-mandatory fields of the input model are automatically filled in with default/initial values. Responsible to post a runtime LGI after update, with field CREATION. It also creates corresponding usSmartLearningAtomInstance structures in the DB, if not already existent.		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if the input JSON structure is wrong OR no input was provided OR the input structure was not correct OR the LGI already exists.		
	HTTP 404 status code if a creation error failed to insert the structure in the DB.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Input Model** **(*)Marks mandatory fields for the input to be accepted**

	-	-	{ "_id" : "{lgi_id}" "mode" : "NORMAL", *"lg_id" : "{lg_id}", *"lg_name" : "{lg_name}", *"learner_id" : "{learner_id}", "created" : "{date/time}", "last_modified" : "{date/time}", *"vertices" : [{ "weight" : "{vertex_weight}", *"label" : "{vertex_name}", *"type" : "SLA", *"_id" : "{vertex_id}"), "_type" : "vertex" }, { "weight" : "{vertex_weight}", *"label" : "{vertex_name}", *"type" : "LEARNING_GOAL", "_id" :"{vertex_id}"), "_type" : "vertex" }, ….], *"edges" : [{ *"weight" : "{edge_weight}", "_id" : "{enumeration}", "_type" : "edge", *"_outV" : "{outward_vertex_id}", *"_inV" : "{inward_vertex_id}", "_label" : "default" }, ….] }

Table 17: LG Open API - GET api/lg/postLGI

URL pattern	**POST api/lg/postLGI/rtm**
Method	POST
Content type	Application/JSON
Description	Create or update a personalised runtime LG instance for a given learner on the MaTHiSiS DB under the usLearningGraphInstance_rtm collection. It also creates corresponding usSmartLearningAtomInstance structures in the DB.
Responses	HTTP 200 status code if successful

	HTTP 400 status code if the input JSON structure is wrong OR no input was provided OR the input structure was not correct OR the runtime LGI already exists.		
	HTTP 404 status code if a creation error failed to insert the structure in the DB.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Input Model** **(*)Marks mandatory fields for the input to be accepted**
			{ "_id" : "{lgi_id}" "mode" : "NORMAL", *"lgi_id" : "{lgi_id}", *"lg_id" : "{lg_id}", *"lg_name" : "{lg_name}", *"learner_id" : "{learner_id}", *"session_id" : "{session_id}", "type " : "{type option}", "created" : "{date/time}", "last_modified" : "{date/time}", *"vertices" : [{ "weight" : "{vertex_weight}", *"label" : "{vertex_name}", *"type" : "SLA", *"_id" : "{vertex_id}"), "_type" : "vertex" }, { "weight" : "{vertex_weight}", *"label" : "{vertex_name}", *"type" : "LEARNING_GOAL", "_id" :"{vertex_id}"), "_type" : "vertex" }, ….], *"edges" : [{ *"weight" : "{edge_weight}", "_id" : "{enumeration}", "_type" : "edge", *"_outV" : "{outward_vertex_id}", *"_inV" : "{inward_vertex_id}", "_label" : "default" }, ….] }

Table 18: LG Open API - GET api/lg/postLGI/rtm

URL pattern	**PUT api/lg/putLG**		
Method	PUT		
Content type	Application/JSON		
Description	Create or update an unpersonalised LG on the MaTHiSiS DB under the lcsLearningGraph collection. Automatically detects LG id from input structure. If the LG is new, thus bears no id, it automatically creates a new structure and assigns a new unique id. Missing non-mandatory fields of the input model are automatically filled in with default/initial values. It also creates corresponding lcsSmartLearningAtom structures in the DB, if not already existent.		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if the input JSON structure is wrong OR no input was provided OR the input structure was not correct.		
	HTTP 404 status code if a creation error failed to insert the structure in the DB.		
	HTTP 500 if the JSON structure of the list failed to be constructed		
Parameters	**Name**	**URL pattern**	**Input Model** **(*)Marks mandatory fields for the input to be accepted**

	-	-	{ "_id" : "{lg_id}" "mode" : "NORMAL", *"lg_name" : "{lg_name}", "lg_descr" : "{lg description}", *"creator_id" : "{tutor_id}", "created" : "{date/time}", "last_modified" : "{date/time}", *"vertices" : [{ * "label" : "{vertex_name}", *"type" : "SLA", *"_id" : "{vertex_id}"), "_type" : "vertex" }, *"label" : "{vertex_name}", *"type" : "LEARNING_GOAL", "_id" :"{vertex_id}"), "_type" : "vertex" }, ….], *"edges" : [{ *"weight" : "{edge+weight}", "_id" : "{enumeration}", "_type" : "edge", *"_outV" : "{outward_vertex_id}", *"_inV" : "{inward_vertex_id}", "_label" : "default" }, ….] }

Table 19: LG Open API - GET api/lg/putLG

URL pattern	**PUT api/lg/putLGI**
Method	PUT
Content type	Application/JSON
Description	Create or update a personalised LG instance on the MaTHiSiS DB under the usLearningGraphInstance collection. Automatically detects LGI id from input structure. If the LGI is new, thus bears no id, it automatically creates a new structure and assigns a new unique id. Missing non-mandatory fields of the input model are automatically filled in with default/initial values. It also creates corresponding usSmartLearningAtomInstance structures in the DB, if

<table>
<tr><td></td><td colspan="3">not already existent.</td></tr>
<tr><td>Responses</td><td colspan="3">HTTP 200 status code if successful</td></tr>
<tr><td></td><td colspan="3">HTTP 400 status code if the input JSON structure is wrong OR no input was provided OR the input structure was not correct.</td></tr>
<tr><td></td><td colspan="3">HTTP 404 status code if a creation error failed to insert the structure in the DB.</td></tr>
<tr><td></td><td colspan="3">HTTP 500 if the JSON structure of the list failed to be constructed</td></tr>
<tr><td>Parameters</td><td>Name</td><td>URL pattern</td><td>Input Model
(*)Marks mandatory fields for the input to be accepted</td></tr>
<tr><td></td><td>-</td><td>-</td><td>{
"_id" : "{lgi_id}"
"mode" : "NORMAL",
*"lg_id" : "{lg_id}",
*"lg_name" : "{lg_name}",
*"learner_id" : "{learner_id}",
"created" : "{date/time}",
"last_modified" : "{date/time}",
*"vertices" : [
{
"weight" : "{vertex_weight}",
*"label" : "{vertex_name}",
*"type" : "SLA",
*"_id" : "{vertex_id}"),
"_type" : "vertex"
},
{
"weight" : "{vertex_weight}",
*"label" : "{vertex_name}",
*"type" : "LEARNING_GOAL",
"_id" :"{vertex_id}"),
"_type" : "vertex"
},
….
],
*"edges" : [
{
*"weight" : "{edge_weight}",
"_id" : "{enumeration}",
"_type" : "edge",
*"_outV" :
"{outward_vertex_id}",
*"_inV" : "{inward_vertex_id}",
"_label" : "default"
},
….
]
}</td></tr>
</table>

Table 20: LG Open API - GET api/lg/putLGI

URL pattern	DELETE api/lg/deleteLGs	
Method	DELETE	
Description	Delete (drop) all LGs under the lcsLearningGraph collection in the MaTHiSiS DB	
Responses	HTTP 200 status code if successful	
Parameters	Name	URL pattern
	-	-

Table 21: LG Open API - GET api/lg/deleteLGs

URL pattern	DELETE api/lg/deleteLGIs	
Method	DELETE	
Description	Delete (drop) all LGIs under the usLearningGraphInstance collection in the MaTHiSiS DB. If a learner id is defined, delete only the LGIs that this learner has worked on. If a LG id is defined, delete only the LGIs instantiating this LG.	
Responses	HTTP 200 status code if successful	
Parameters	Name	URL pattern
	uid lgid	?uid={learner_id} &lgid={lg_id}

Table 22: LG Open API - GET api/lg/deleteLGIs

URL pattern	DELETE api/lg/deleteLGIs/rtm	
Method	DELETE	
Description	Delete (drop) all runtime LGIs under the usLearningGraphInstance_rtm collection in the MaTHiSiS DB. If a learner id is defined, delete only the runtime LGIs that this learner has worked on. If a LGI id is defined, delete only the runtime LGIs pertaining to this LGI. If a LG id is defined, delete only the runtime LGIs pertaining to this LG. If a session id is defined, delete only the runtime LGIs created during that session.	
Responses	If Success HTTP 200 status code (OK)	
	If Error HTTP 304 status code (Not Modified)	
Parameters	Name	URL pattern

	uid lgid lgiid sid	?uid={learner_id} &lgid={lg_id} &lgiid={lgi_id} &sid={session_id}

Table 23: LG Open API - GET api/lg/deleteLGIs/rtm

URL pattern	**DELETE api/lg/deleteLG**	
Method	DELETE	
Description	Delete (drop) a specific LG in the lcsLearningGraph collection in the MaTHiSiS DB	
Responses	HTTP 200 status code if successful	
	HTTP 400 status code if no SLA ID was provided	
Parameters	**Name**	**URL pattern**
	id	?id={lg_id}

Table 24: LG Open API - GET api/lg/deleteLG

URL pattern	**DELETE api/lg/deleteLGI**	
Method	DELETE	
Description	Delete (drop) a specific LG in the usLearningGraphInstance collection in the MaTHiSiS DB	
Responses	HTTP 200 status code if successful	
	HTTP 400 status code if no SLA ID was provided	
Parameters	**Name**	**URL pattern**
	id	?id={lgi_id}

Table 25: LG Open API - GET api/lg/deleteLGI

URL pattern	**DELETE api/lg/deleteLGI/rtm**
Method	DELETE
Description	Delete (drop) a specific LG in the usLearningGraphInstance_rtm collection in the MaTHiSiS DB
Responses	HTTP 200 status code if successful

	HTTP 400 status code if no SLA ID was provided	
Parameters	**Name**	**URL pattern**
	id	?id={lgi_rtm_id}

Table 26: LG Open API - GET api/lg/deleteLGI/rtm