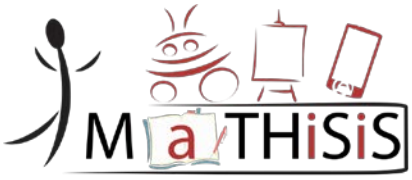

Managing Affective-learning THrough Intelligent atoms and Smart InteractionS

D3.2 The MaTHiSiS Smart Learning Atoms

Workpackage	WP3 – Smart Learning Atoms and Graph Tools
Editor(s):	Thomas TECHENE, DXT Dorothea TSATSOU, Anastasios DIMANIDIS, Nicholas VRETOS, CERTH
Responsible Partner:	DXT
Quality Reviewers	Ana Luiza PONTUAL, Ana Piñuela ATOS Andy BURTON, NTU
Status-Version:	Final v1.0
Date:	Project Start Date: 1/1/2016; Duration: 36 months Deliverable Due Date: 31/12/2017 Submission Date: 15/12/2017
EC Distribution:	PU
Abstract:	This document constitutes the second and final version of the Smart Learning Atoms (SLA) report of the MaTHiSiS project. The SLA, which is central to the approach taken in the MaTHiSiS project, is an atomic and independent piece of knowledge or skill that will allow the users of the MaTHiSiS system to re-use their work in other similarly structured learning scenarios, as is often the case in modern schools or work environments.
Keywords:	Smart Learning Atom, atomicity, reuseability, individuality, non-linearity, Open API
Related Deliverable(s)	*D2.2 - Full scenarios of all use cases (M9)* *D2.4 - Full system architecture (M15)*

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

	D3.1 The MaTHiSiS Smart Learning Atoms *D3.4 - The MaTHiSiS Learning Graphs* *D3.6 - Experience Engine* *D3.9 - MaTHiSiS Frontend Components* *D6.2 - The MaTHiSiS Learning Graph Engine* *D6.4 - Synchronous and Asynchronous collaboration among platform agents* *D7.1 - Integration Strategy and planning* *D7.3 - MaTHiSiS platform, 2nd release*

Document History

Version	Date	Change editors	Changes
0.1	15/11/2017	Thomas TECHENE (DXT)	Initial version of the document.
0.2	04/12/2017	Thomas TECHENE (DXT)	Contribution Section 1, 3.3, 4, 5
0.3	10/12/2017	Dorothea Tsatsou, Anastasios DIMANIDIS, Nicholas VRETOS (CERTH)	Contribution Section 2, 3, 6, 7
0.4	13/12/2017	Thomas TECHENE (DXT)	Document alignment
0.5	14/12/2017	Andy Burton (NTU)	Internal Review
0.6	14/12/2017	Thomas TECHENE (DXT)	Changes after internal review
0.7	15/12/2017	Ana Luiza Pontual (ATOS)	Internal Review
0.8	15/12/2017	Thomas TECHENE (DXT)	Changes after internal review
0.9	15/12/2017	Ana Piñuela (ATOS)	Final quality review
1.0	15/12/2017		FINAL VERSION TO BE SUBMITTED

Table of Contents

Document History 3
Table of Contents 4
List of Tables 5
List of Figures 6
List of Acronyms 7
Project Description 8
Executive Summary 9
1. Introduction 11
2. Smart Learning Atoms 12
2.1 Objectives and definitions 12
2.2 Methodology and dependencies 12
2.3 Smart Learning Atoms educational attributes: concrete examples 14
2.3.1 Atomicity & self-sustainability 14
2.3.2 Re-usability 15
2.3.3 Individuality 18
2.3.4 Non-linearity 18
3. Smart Learning Atom library implementation details 19
3.1 Functionalities 19
3.2 Open API 20
3.3 Interface with the Front-end 22
3.3.1 SLAs in the Learning Content Editor 22
4. Conclusion 26
5. References 27
6. Appendix I: SLA Data Structures 28
7. Appendix I: SLA lib Open API documentation 31

List of Tables

Table 1: Definitions, Acronyms and Abbreviations 7
Table 2: Learning Graph "PE_MEC_12-13" used by partners PE in assisted pilots for mainstream case children aged 12-13 16
Table 3: Learning Graph "JCYL_ASC_PMLD" used by partners JCYL in assisted pilots for learners with autism and PMLD 18
Table 4: Unpersonalised SLA data structure (Collection lcsSmartLearningAtom) 28
Table 5: Personalised SLAI data structure (US_SmartLearningAtomInstance) 29
Table 6: Personalised SLAI runtime record data structure (usSmartLearningAtomInstance_rtm) 30
Table 7: SLA Open API - GET api/sla/getSLAs 31
Table 8: SLA Open API - GET api/sla/getSLA 32
Table 9: SLA Open API - GET api/sla/getLAs 32
Table 10: SLA Open API - GET api/sla/getSLAIs 33
Table 11: SLA Open API - GET api/sla/getSLAI 33
Table 12: SLA Open API - GET api/sla/getSLAIs/rtm 34
Table 13: SLA Open API - GET api/sla/getSLAI/rtm 35
Table 14: SLA Open API - POST api/sla/postSLA 35
Table 15: SLA Open API - POST api/sla/postSLAI 36
Table 16: SLA Open API - POST api/sla/postSLAI/rtm 37
Table 17: SLA Open API - POST api/sla/updateSLAIweight 37
Table 18: SLA Open API - PUT api/sla/putSLA 38
Table 19: SLA Open API - PUT api/sla/putSLAI 39
Table 20: SLA Open API - DELETE api/sla/deleteSLAs 39
Table 21: SLA Open API - DELETE api/sla/deleteSLAIs 39
Table 22: SLA Open API - DELETE api/sla/deleteSLAIs/rtm 40
Table 23: SLA Open API - DELETE api/sla/deleteSLA 40
Table 24: SLA Open API - DELETE api/sla/deleteSLAI 40
Table 25: SLA Open API - DELETE api/sla/deleteSLAI/rtm 41

List of Figures

Figure 1: SLAs and SLA Instances and dependencies with other MaTHiSiS collections 14
Figure 2: Learning Graph "PE_ASC_PMLD" used by partners PE in assisted pilots for learners with autism and PMLD 15
Figure 3: The base URL and GET methods available through the SLA lib Open API 20
Figure 4: The DELETE, POST and PUT methods available through the SLA lib Open API 21
Figure 5: UI mock-up for the SLA Editor 23
Figure 6: First Prototype of the SLA Editor 23
Figure 7: Current state of SLA Editor 24

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
LCE	Learning Content Editor
LCM	Learning Content Manager
LCS	Learning Content Space
LES	Learning Experience Supervisor
LG	Learning Graph
LGR	Learning Graph Repository
MEC	Mainstream Education Case
PA	Platform Agent
PMLDC	Profound and Multiple Learning Disabilities Case
SLA	Smart Learning Atom
SLA	Smart Learning Atom Instance
UI	User Interface
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 goals for 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 [1] which can be learned and assessed in a single, short-term iteration, targeting certain problems. More than one SLA, working together on the same 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

This deliverable constitutes the second and final version of the Smart Learning Atoms report of the MaTHiSiS project and presents the Smart Learning Atom (SLA), one of the key concepts in MaTHiSiS.

Smart Learning Atoms (SLAs) are atomic and complete pieces of knowledge, competence and/or skills, which can be learned and assessed in a single, short-term learning process iteration from a learner. 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 an educational process.

SLAs comprise the basis of the novel learning approach of MaTHiSiS. Their standalone and self-contained nature allows learning scenario designers to define re-usable, versatile learning entities, applicable to various learning subjects, which can contribute to more complex/compound learning goals. SLAs are the constructive elements of Learning Graphs (cf. Deliverable 3.4 *The MaTHiSiS Learning Graphs*), where already existing or new SLAs are connected to composite learning goals, thus constructing a relational network of the learning scenario objectives.

In this relational network, the relations are directed, denoting that one or more network nodes (nodes standing for knowledge, skills, competences) contribute to the apprehension of other nodes. Since SLAs represent **atomic** and **standalone** competences, they are never related to each other in any Learning Graph structure, since no further divisible units can contribute to an SLA. They are rather always contributing to learning goals.

Besides atomicity, another important attribute of Smart Learning Atoms is **re-usability**. In this sense, Learning Experience enablers (i.e. teachers/caregivers/trainers) can engineer their desired SLAs from scratch, but they can also draw from a library of pre-existing SLAs maintained in the MaTHiSiS platform, and enrich them with additional custom attributes.

Furthermore, breaking down learning objectives to fundamental sub-components (SLAs) allows for **non-linear** and **highly adaptive discretisation** of the learning process, which does not have to follow a rigid collective and cascading style anymore, but rather adopts a non-linear relational operational scheme, i.e. the organization of the Learning Experience in Learning Graphs. More specifically, the hierarchical composition of the learning objectives from atomic (SLAs) to composite (learning goals) units enables the learning experience to alternate its focus on mastering each atomic learning content constituent (i.e. the SLAs), which in turn implies mastering the composite learning goals.

Non-linear alternation in training SLAs during the learning process will ensure a highly personalized adaptation scheme, based on the learner's particular apprehension abilities, learning style and uptake over the knowledge/skills to acquire, which at the same time ensures the maintenance of the engagement of the learners in the learning process.

This learner-centric scheme is also supported by the ability of the MaTHiSiS Platform Agents (PAs), acting as learning process facilitators, to deploy precise learning activities in each SLA for each individual learner taking part in a learning scenario, in each iteration of the learning process. To this end, each SLA is attached to one or more generic Learning Actions (LAs), which the MaTHiSiS system can materialise in different ways on the different PAs (cf. D3.6 *Experience Engine[2]*).

The **SLA lib** and its accompanying **Open API** is the library responsible for the creation, manipulation and deletion of the three SLA structure manifestations (the SLA, SLAI and runtime SLAI). This component is namely the **SLA lib** and its accompanying **Open API**. The SLA lib has only been slightly modified since the first iteration of this document, but will be described here for self-sustainability purposes of the document, while the Open API has been enhanced with service filters and improvements.

The SLAs are key in several places of the MaTHiSiS Front-end:

- In the Learning Content Editor (LCE), all the SLAs created by MaTHiSiS users, stored in the Learning Graphs Repository (LGR), can be browsed, viewed and edited.
- The LCE is where new SLAs can be created and then published, and also existing ones can be edited by tutors.
- In the LCE, SLAs can also be used in Learning Graphs being created by tutors.
- Finally, the Learning Experience Supervisor allows both tutors and learners to examine the relevant SLAs and the graphs to which they belongs involved in their Learning Experience.

1. Introduction

D3.2 - The MaTHiSiS Smart Learning Atoms is the second and final report of the Smart Learning Atoms. This concept is central to the approach taken in the MaTHiSiS project and one of the outcomes of work package *WP3 - Smart Learning Atoms and Graph Tools* and more specifically *T3.1 - Educational Content development (Smart Learning Atoms)*.

This document is organized as follows:

- Section 1 introduces the purpose and structure of this deliverable.
- Section 2 describes the Smart Learning Atoms and how they are integrated in the MaTHiSiS platform. Furthermore examples of Smart Learning Atoms are provided to understand the rationale behind this concept.
- Section 3 goes through the particulars implementations of the library that is responsible for the creation, manipulation and deletion of the three SLA structure manifestations (the SLA, SLAI and runtime SLAI) and describes the interface of the SLA in the MaTHiSiS front end.
- Section 4 presents the conclusions of the document.

2. Smart Learning Atoms

This section describes the MaTHiSiS core educational process innovation, namely the Smart Learning Atoms (SLAs).

2.1 Objectives and definitions

Smart Learning Atoms (SLAs) are atomic and complete pieces of knowledge, competence and/or skills, which can be learned and assessed in a single, short-term learning process iteration from a learner. 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 an educational process.

SLAs comprise the basis of the novel learning approach of MaTHiSiS. Their standalone and self-contained nature allows learning scenario designers to define re-usable, versatile learning entities, applicable to various learning subjects, which can contribute to more complex/compound learning goals. SLAs are the constructive elements of Learning Graphs (cf. Deliverable 3.4 *The MaTHiSiS Learning Graphs*), where already existing or new SLAs are connected to composite learning goals, thus constructing a relational network of the learning scenario objectives.

In this relational network, the relations are directed, denoting that one or more network nodes (nodes standing for knowledge, skills, competences) contribute to the apprehension of other nodes. Since SLAs represent **atomic** and **standalone** competences, they are never related to each other in any Learning Graph structure, since no further divisible units can contribute to an SLA. They are rather always contributing to learning goals.

Besides atomicity, another important attribute of Smart Learning Atoms is **re-usability**. In this sense, Learning Experience enablers (i.e. teachers/caregivers/trainers) can engineer their desired SLAs from scratch, but they can also draw from a library of pre-existing SLAs maintained in the MaTHiSiS platform, and enrich them with additional custom attributes.

Furthermore, breaking down learning objectives to fundamental sub-components (SLAs) allows for **non-linear** and **highly adaptive discretisation** of the learning process, which does not have to follow a rigid collective and cascading style anymore, but rather adopts a non-linear relational operational scheme, i.e. the organization of the Learning Experience in Learning Graphs. More specifically, the hierarchical composition of the learning objectives from atomic (SLAs) to composite (learning goals) units enables the learning experience to alternate its focus on mastering each atomic learning content constituent (i.e. the SLAs), which in turn implies mastering the composite learning goals.

Non-linear alternation in training SLAs during the learning process will ensure a highly personalized adaptation scheme, based on the learner's particular apprehension abilities, learning style and uptake over the knowledge/skills to acquire, which at the same time ensures the maintenance of the engagement of the learners in the learning process.

This learner-centric scheme is also supported by the ability of the MaTHiSiS Platform Agents (PAs), acting as learning process facilitators, to deploy precise learning activities in each SLA for each individual learner taking part in a learning scenario, in each iteration of the learning process. To this end, each SLA is attached to one or more generic Learning Actions (LAs), which the MaTHiSiS system can materialise in different ways on the different PAs (cf. D3.6 *Experience Engine[2]*).

2.2 Methodology and dependencies

In order to facilitate the self-contained nature of Smart Learning Atoms and at the same time their adaptability to different learner specifications, SLAs will take up two forms in the MaTHiSiS learning setting.

1. **Unpersonalised, core SLAs** are maintained independently of Learning Graphs (LG) that they might take part in. This allows SLAs to be retrievable and re-usable in different learning scenarios (outlined by Learning Graphs). The SLA structure comprises of the universal properties of the particular competences/skills/knowledge pieces they entail. This pertains to a unique identifier in the MaTHiSiS database, a short name and a more detailed description of what each SLA is about, along with a reference to their original creator or last user having modified them and accompanying generation and modification dates. Lastly, they include a reference to the generic, PA-agnostic Learning Actions that may be deployed during the learning process to actively train the encapsulated competences/skills/knowledge that the SLAs represent. These structures reside on the MaTHiSiS Learning Content Space (LCS), as detailed in Deliverable **D7.3** ***MaTHiSiS platform, 2nd release*** [5].
2. **Personalised SLA instances (SLAIs)** are created for each learner, upon the initiation of any learning experience. SLA instances are strictly personal, reflecting the uptake of a learner over a particular competence, piece of knowledge, or skill. The instances incorporate a reference to their corresponding core unpersonalised SLA counterparts and further bear a scalar weight in [0, 1], which indicates the learner's knowledge/skill/competence acquisition progress. Their structure is similar to the unpersonalised SLAs' one, with the difference that instead of a reference to the creator, they refer to the individual learner and with the addition of the competence weight described previously. SLA instances are again maintained independently from their connected learning scenarios (Learning Graphs), and therefore progress over them will affect any running or future scenarios (Learning Graphs), that the learner might be involved in, which contain the particular SLAI. SLAIs reside on the MaTHiSiS User Space (US), detailed in Deliverable **D7.3** ***MaTHiSiS platform, 2nd release*** [5].
3. For learning analytics purposes, a historical record of **runtime SLA instances** per learner is maintained in the MaTHiSiS DB. While long-term personalised SLA instances always reflect the last state of each SLA instance per user, learning analytics require analysing the fluctuations of SLA weights across the course of a learner's experience or across different learning experiences that involve this competence. To this end, a historical record of runtime (denoted as 'SLAI_rtm') instances of SLAs is maintained, bearing a connection to their long-term (last state) SLA instance counterpart for each learner, along with the reference to the particular session that the SLAI was modified in during runtime and the type of process that created them (e.g. creation, personalization, adaptation, etc.). These structures also reside on the MaTHiSiS Cloud Learner Space (CLS).

Evidently, SLAs' most prominent dependency is the Learning Graphs (LGs). Although SLAs can be maintained individually, they cannot be trained unless they take part in at least one concrete learning scenario (which is fulfilled by a LG). LG structures contain a reference to each SLA structure they contain, following the same structural logic as SLAs. I.e. core unpersonalised LGs refer to core unpersonalised SLAs, personal LG instances refer to the corresponding personal SLA and runtime LG instances refer to their respective runtime SLA instances for the particular session.

As mentioned previously, another fundamental dependency of the SLAs, limited to the unpersonalised structures, is the connection to specific, PA-agnostic Learning Actions (LAs). For a SLA to be attained by any learner, it needs to have at least one concrete activity (LA) attached to it, which can train/teach/reinforce the particular SLA on MaTHiSiS PAs. As detailed in ***Deliverable 3.6 Experience Engine*** [2], LAs are conceptual and PA-agnostic activities, but they are materialized practically by suitable Learning Materials on each specific Platform Agent.

Figure 1 graphically illustrates the interdependencies pertaining to SLAs, based on the deployment of the MaTHiSiS database schema. The collections that the SLAs are related to are portrayed as empty placeholders for visual simplification purposes. This schema remains stable since the first version of this document.

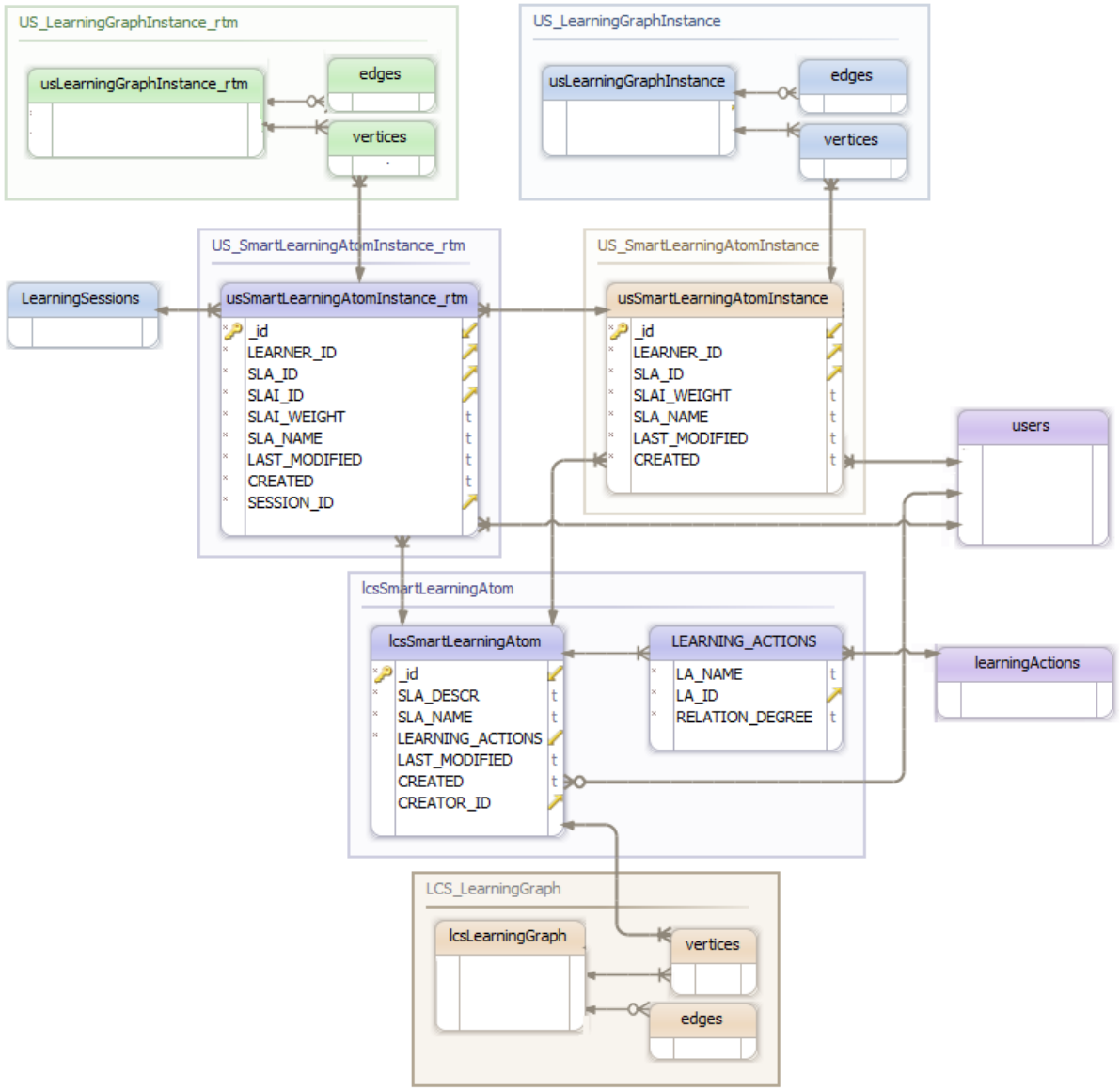

Figure 1: SLAs and SLA Instances and dependencies with other MaTHiSiS collections

2.3 Smart Learning Atoms educational attributes: concrete examples

The following examples outline the core attributes that pertain to the Smart Learning Atoms, as it emerged in the MaTHiSiS assisted pilots.

2.3.1 Atomicity & self-sustainability

Atomicity and self-sustainability refer to the SLAs inherent quality of being standalone and self-sufficient in terms of the knowledge/skills/competences they represent. This is apparent in (a) their indivisible character in learning scenarios, as represented in LGs where SLAs contribute to compose learning goals, but nothing contributes to SLA and (b) their primitiveness, expressed implicitly by their capacity to contribute to more than one learning goals. Figure 2 exemplifies this fact in a LG used by one of the MaTHiSiS partners (ISTITUTO COMPRENSIVO STATALE B. LORENZI FUMANE VR , Italy) for autistic children aged 8-11 years old. All SLAs (in orange) are independent of each other and nothing contributes to them, while goals are commonly contributed to and may contribute to other goals (a), and SLA *Sequence Reproduction* in particular contributes to two goals: *Sequencing* and *Attention Skills* (b). Note *Sequence Reproduction* that participates in the two goals with a different contribution gravity, defined by the edge weight.

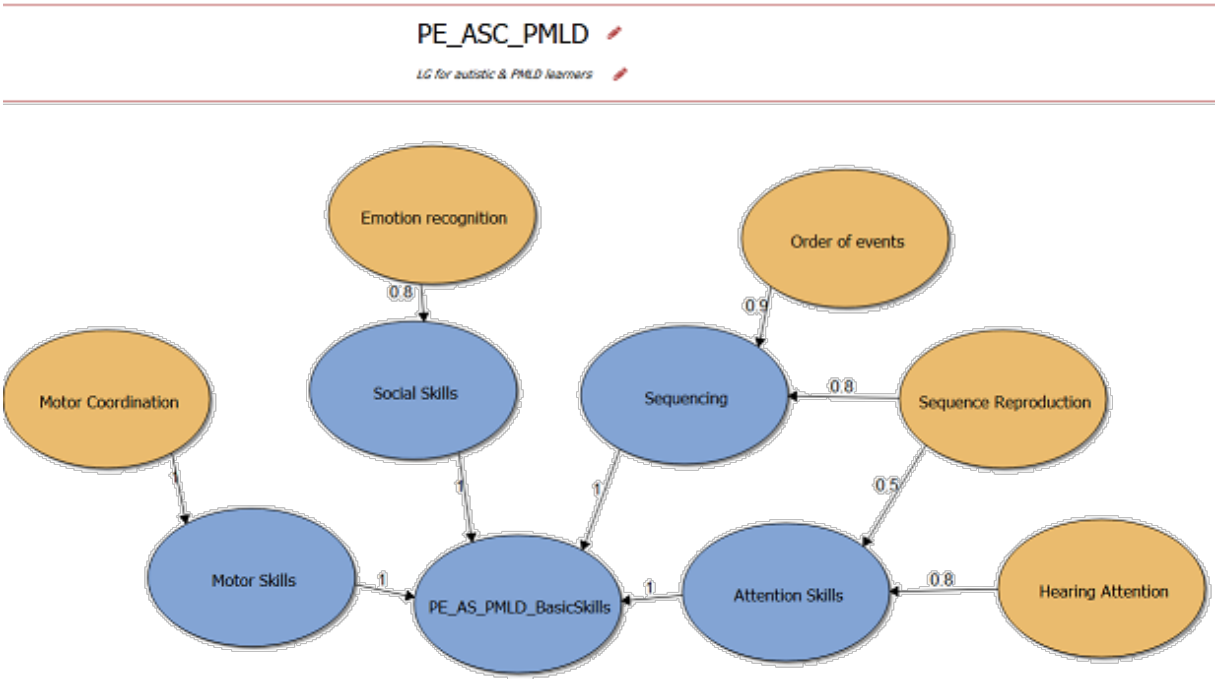

Figure 2: Learning Graph "PE_ASC_PMLD" used by partners PE in assisted pilots for learners with autism and PMLD

2.3.2 Re-usability

Re-usability refers to the SLAs' capacity to be re-used in different learning scenarios (organised in LGs) and be trained in any scenario that uses them. During assisted pilots, 16 SLAs were re-used in 2 to 9 LGs. Table 2 and Table 3 represent the components used in two of the LGs used in pilots, with the shared SLAs denoted in column "Same as". In the LG "PE_MEC_12-13", two out of the total three SLAs are reused in one to eight more LGs during assisted pilots, while in LG "JCYL_ASC_PMLD" all seven SLAs of the LG are reused in one to eight more LGs.

Learning Goal	Weight (0 to 1.0)	Smart Learning Atom	Learning Action	Same as
Attention skills	0.5	Sequence reproduction	Reproduce a sequence of pictures / sounds	**SLA "Sequence reproduction"** *in LG(s)* **PE_ASC_PMLD**
Social skills	0.5	Basic emotion identification	Classify emotions	
	0.8	Emotion recognition	Identify emotional facial expressions	**SLA: "Emotion Recognition"** *in LG(s)*

				JCYL_ASC_PMLD; JCYL_ME_4-6; JCYL_ME_9-12; PE_ASC_PMLD; PE_ME_6-9; PE_AS_6; FMD_LCS FMD_ASC_15-20
			Demonstrate understanding of different emotions	

Table 2: Learning Graph "PE_MEC_12-13" used by partners PE in assisted pilots for mainstream case children aged 12-13

Learning Goal	Weight (0 to 1.0)	Smart Learning Atom	Learning Action	Same as
Social skills	0.8	Emotion recognition	Identify emotional facial expressions	**SLA: "Emotion Recognition"** *in LG(s)* **JCYL_ME_9-12; JCYL_ME_4-6; PE_ASC_PMLD; PE_ME_6-9; PE_AS_6; PE_ME_12-13; FMD_LCS FMD_ASC_15-20**
			Demonstrate understanding of different emotions	
Navigation	0.9	Left and right identification	Identify left and right (own/object)	**SLA: "Left and right identification"** *in LG(s)* **UoN_ASD_PMLD**
			Recognise left and right direction	
			Turn left and right	

	0.7	Area recognition	Match name or symbol to different rooms	**SLA: "Area recognition"** *in LG(s)* **UoN_ASD_PMLD**
	0.8	Targeted location navigation	Walk/navigate to find location	**SLA: "Targeted location navigation"** *in LG(s)* **UoN_ASD_PMLD**
Sequencing	0.9	Sorting	Sort objects into right order	**SLA: "Sorting"** *in LG(s)* **UoN_ASD_PMLD; PE_ASC_PMLD**
			Sort ascending / descending on one dimension (e.g. height, number in group)	
	0.8	Order of events	Sort pictures into logical order e.g. child waking up, dressing, eating breakfast, leaving house. Identify incorrect sequences	**SLA: "Order of events"** *in LG(s)* **UoN_ASD_PMLD**
			Sort words in sentence into logical order. Identify incorrect order.	

<table>
<tr><td rowspan="3">Vocabulary</td><td rowspan="2">0.8</td><td rowspan="2">Object recognition</td><td>(Correctly) point to picture of object named by tutor, being given an increasing number of objects to choose</td><td rowspan="3">SLA: "Objecr Recognition"

in LG(s)

UoN_ASD_PMLD;</td></tr>
<tr><td>Match written name to picture being given an increasing number of objects to choose from</td></tr>
<tr><td>Fish or pairs</td></tr>
<tr><td>0.5</td><td>Area recognition</td><td colspan="2">SLA shared with "Navigation", but with a smaller relation weight to this goal. What this connection means is that when this SLA is trained, it will mostly train Navigation, but it will "rub off" a bit for Vocabulary as well.</td></tr>
</table>

Table 3: Learning Graph "JCYL_ASC_PMLD" used by partners JCYL in assisted pilots for learners with autism and PMLD

2.3.3 Individuality

Individuality refers to the capacity of the SLAs to be instantiated to strictly personal structures (Smart Learning Atom Instances) that carry each learner's personal level of achievement in the specific piece of knowledge/skill/competence they represent.

For instance, the SLA *Emotion Recognition*, which was widely used in the assisted pilot, as it was reused in nine Learning Graphs (cf. the previous section), was instantiated in 142 personal SLAIs for different learners of the platform. An anonymous learner of the Spanish pilot, pulled randomly from the MaTHiSiS DB based on correlation with their learning environment, worked on nine different SLAs, contained in two different LGs, as identified through the learner's personal SLAIs stored in the database. In total, around 420 SLAIs were created for all learners that participated in the assisted pilots.

2.3.4 Non-linearity

Non-linearity pertains to the ability of the SLAs to be trained in no particular order but based solely on the individual performance and affect state of the learners during the execution of a learning experience. This will be further detailed in Deliverable D3.4[10], as it pertains prominently to the relational structure of the Learning Graphs that enable this capacity to the SLAs.

3. Smart Learning Atom library implementation details

This section goes through the particulars implementations of the library that is responsible for the creation, manipulation and deletion of the three SLA structure manifestations (the SLA, SLAI and runtime SLAI), as described in Section 2.2. This component is namely the **SLA lib** and its accompanying **Open API**. The library has only been slightly modified since the first iteration of this document, but will be described here for self-sustainability purposes of the document, while the Open API has been enhanced with service filters and improvements. The details of the SLA, SLAI and runtime SLAI data structures are described in Appendix I: SLA Data Structures.

3.1 Functionalities

The SLA library incorporates all the methods and functionalities required to create, access and modify the SLA, SLAI and runtime SLAI data structures. In this library, long-term SLAI structures and runtime snapshots of SLAIs are treated as the same structure, applicable to different serialisation on the MaTHiSiS database through the SLA lib Open API (detailed in Section 3.2). It is implemented as a Java library, which is embedded to the Web Service that implements the Open API. More specifically, the library offers functionalities to:

- Create an unpersonalised SLA, 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 5).
- Create a personal SLA 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 SLA instance, based on a given JSON input of a serialised SLAI (data structure as per Section 6) .
- 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 SLAIs, initial default weight (0.3) is set in the very first instantiation of an SLA to a personal SLAI, unless explicitly stated otherwise.
- 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 personalisation and adaptation process, a direct functionality to update SLAI weights is exposed, which allows to set new SLAI weights without having to explicitly retrieve and modify the entire SLAI structure.
- Similarly, for facilitating the SLA editing process, a direct functionality to insert and remove learning actions from SLAs is exposed, without having to explicitly retrieve and pass the entire LA structure.
- Evaluate validity of SLA, SLAI and runtime SLAI structures as per the mandatory fields.
- Create JSON serialisations for each of the supported data structures (SLA, SLAI, runtime SLAI) to be inserted to the MaTHiSiS DB through the SLA lib Open API.

3.2 Open API

The *JAX-RS*[1] Java API for RESTful Web Services was used to create a Web Service, exposing the functionalities of the SLA lib according to the Representational State Transfer (REST) architectural pattern. The API is also responsible for serialising, retrieving and deleting SLA, SLAI and runtime SLAI entries to and from the MaTHiSiS database. While SLA lib is in charge of processing (access, creation, update) of SLA and SLAI 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 SLA lib Open API is available through the central *<MaTHiSiS base URL>/api/sla/* base URL. MaTHiSiS components that consume SLA 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 SLA lib Open API, are listed below in Figure 3 and Figure 4 are described in Appendix I: SLA lib Open API documentation.

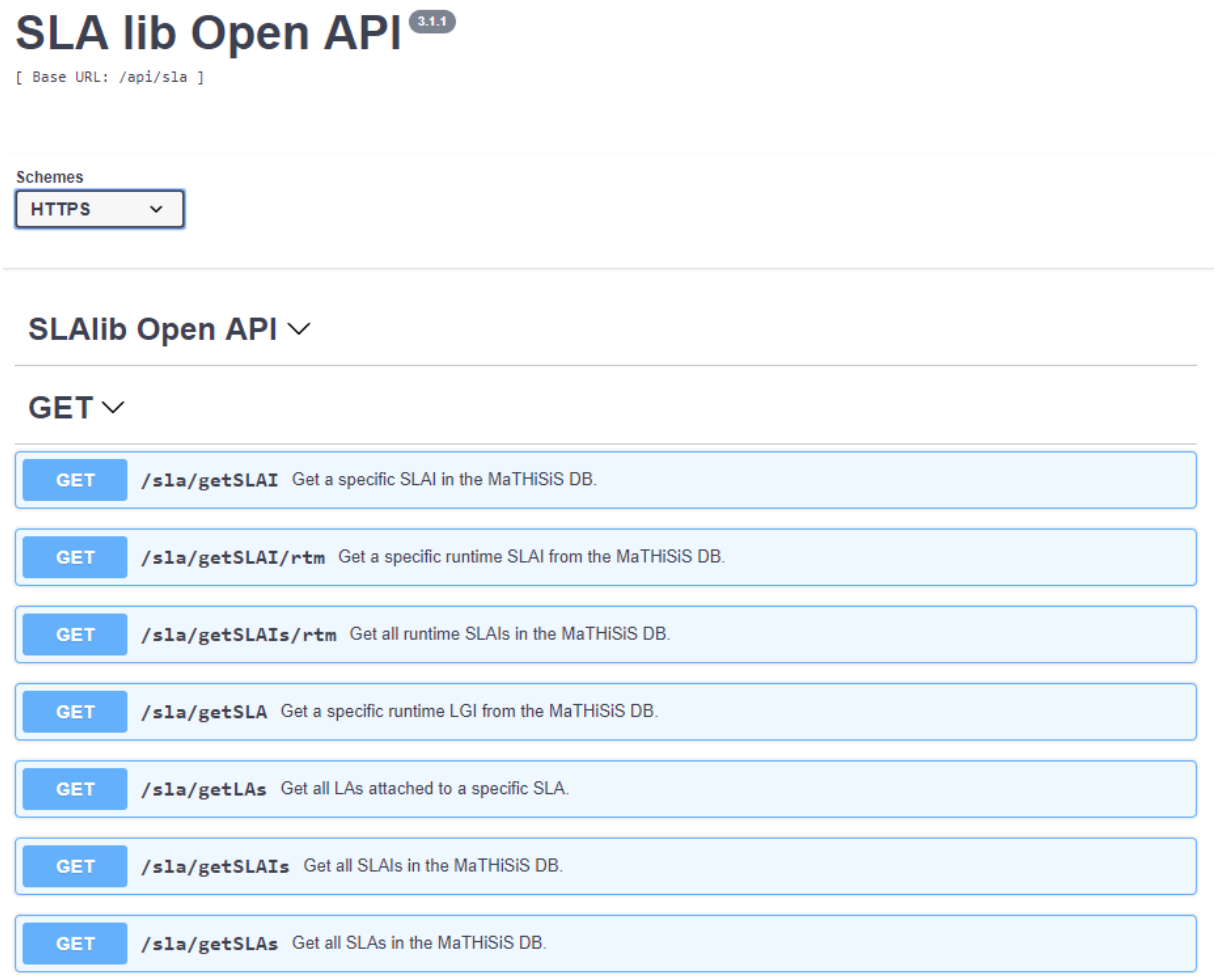

Figure 3: The base URL and GET methods available through the SLA lib Open API

[1] https://jax-rs-spec.java.net/

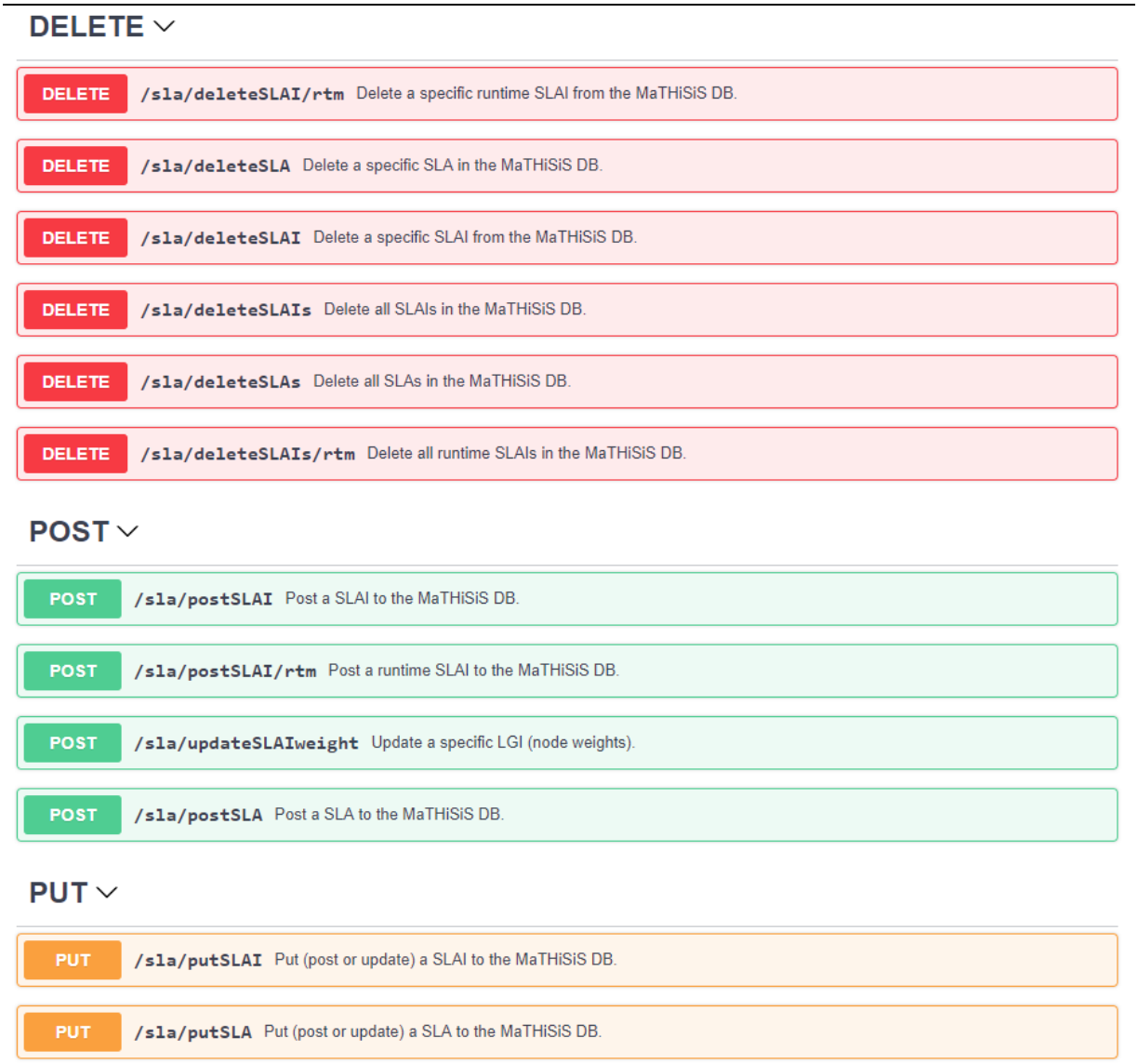

Figure 4: The DELETE, POST and PUT methods available through the SLA lib Open API

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

- Retrieve all SLAs, SLAIs or runtime SLAIs from the MaTHiSiS database. For SLAIs and runtime SLAIs, filters to retrieve instances specific to a particular core unpersonalised SLA or a particular user are available.
- Retrieve a specific SLA, SLAI or runtime SLAI from the MaTHiSiS database.
- Retrieve all LAs attached to a particular SLA. This is an assistive functionality for back-end components, through which the LAs can be retrieved directly, without having to access the entire SLA structure and traverse it to retrieve attached LAs.
- Delete all SLAs, SLAIs or runtime SLAIs from the MaTHiSiS database. For SLAIs and runtime SLAIs, filters to delete instances specific to a particular core unpersonalised SLA or a particular user are available.
- Delete a specific SLA, SLAI or runtime SLAI from the MaTHiSiS database.
- Post (insert) a SLA, SLAI or runtime SLAI 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 SLA, SLAI, runtime SLAI structure models, per case.
- Update a SLAI's weight. This is an assistive functionality, used in the back-end by the DSS and the LGE (cf. Deliverables D6.2 and D6.4 [7][8]), where the SLAI weight can be updated directly,

without having to access the entire SLAI structure, traverse and alter it outside of the control of the SLA lib. This functionality also evokes the insertion of a runtime SLAI entry in DB each time it is called.

- Put (insert or update) a SLA, SLAI or runtime SLAI to the MaTHiSiS database. These functionalities implement structure validation, to ensure that the structure to be inserted/updated complies with the defined SLA, SLAI, runtime SLAI structure models, per case.

3.3 Interface with the Front-end

Smart Learning Atoms are core elements in the pedagogical methodology introduced in MaTHiSiS, and they are key in several places of the MaTHiSiS Front-end:

- In the Learning Content Editor (LCE), all the SLAs created by MaTHiSiS users, stored in the Learning Graphs Repository (LGR), can be browsed, viewed and edited.
- The LCE is where new SLAs can be created and then published, and also existing ones can be edited by tutors.
- In the LCE, SLAs can also be used in Learning Graphs being created by tutors.
- Finally, the Learning Experience Supervisor allows both tutors and learners to examine the relevant SLAs and the graphs to which they belongs involved in their Learning Experience.

3.3.1 SLAs 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. SLAs management through LCE can be broken down in four main objectives:

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

Multiple users can take part during the content creation process in MaTHiSiS, working on the different building blocks (LG, SLA, LA, LAM and LM). Manipulating all these concepts in a single, unified edition tool would potentially lead to a difficult and non user-friendly tool. An early decision in the project was taken to create separate and independent tools for each, with the remaining needs of interconnection between these editing tools. Furthermore, quick navigation between the tools is mandatory, making the work on different elements during the same content creation process more pleasant.

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

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

A description of the entire Learning Content Editor is provided in the ***D3.9 – MaTHiSiS Frontend Components*** **[9]**.

In the 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. The approach taken for UI design was to work on mock-ups for the entire tool-set(LG Editor, SLA Editor, LA Editor, LAM 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 SLA Editor

The SLA editor has evolved over the time. Some screenshot below illustrate the different steps the editor has been through.

Figure 5 shows the UI mock-up for the SLA Editor tool designed in the 1st year:

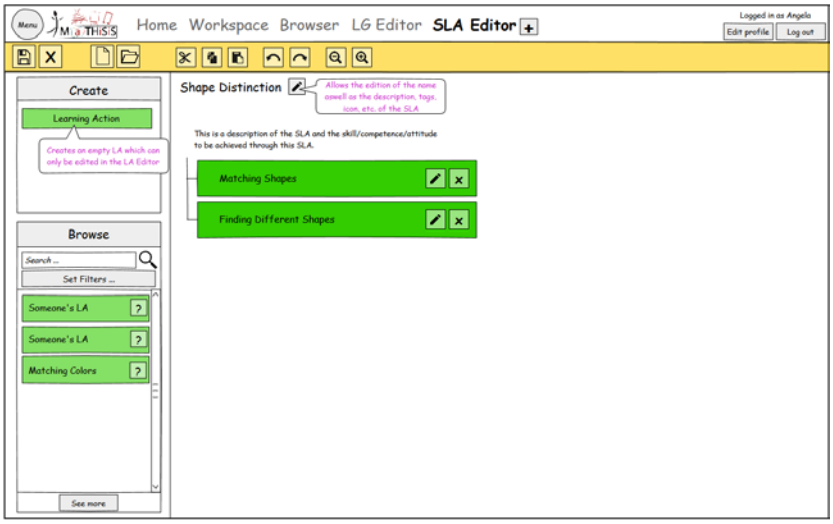

Figure 5: UI mock-up for the SLA Editor

Figure 6 shows the first prototype of the SLA Editor developed during the 1st year:

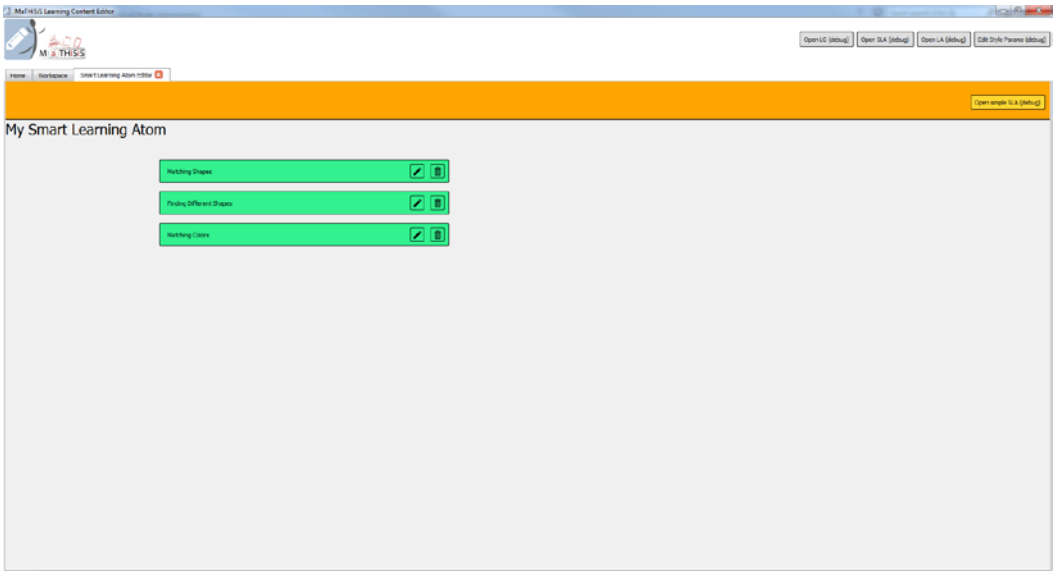

Figure 6: First Prototype of the SLA Editor

Figure 7 shows the current status of the SLA Editor after the 2nd year.

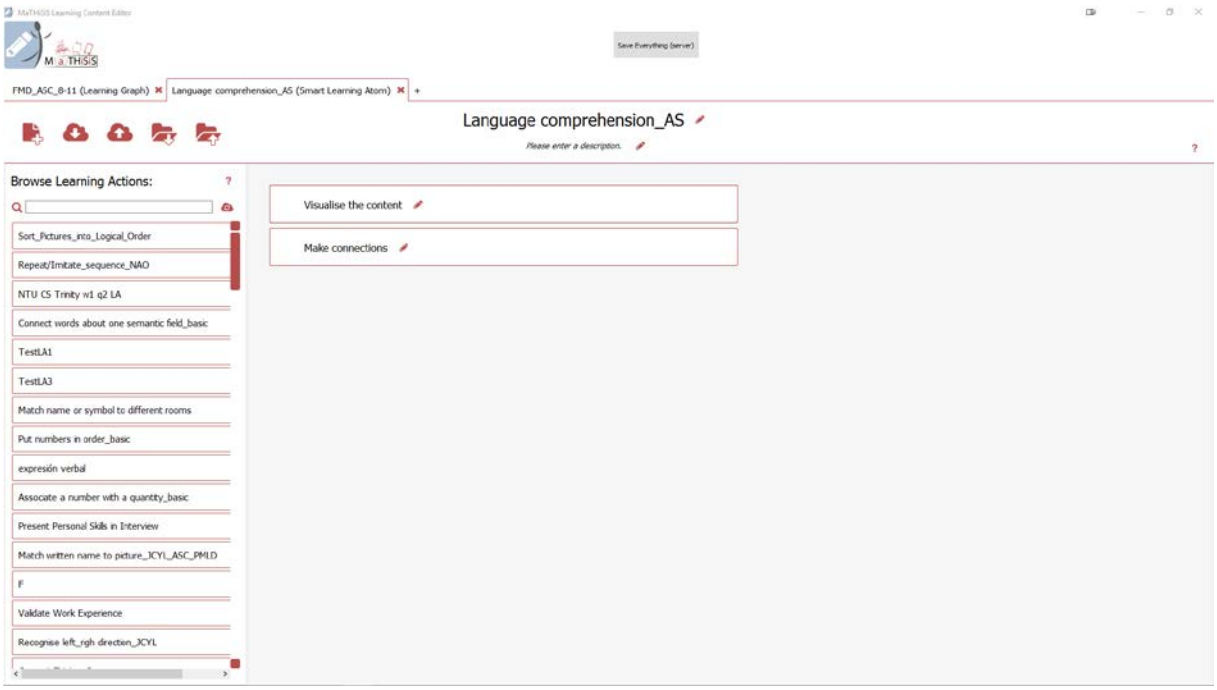

Figure 7: Current state of SLA Editor

The SLA content is represented by a list of Learning Actions, in no specific order (although the list can be rearranged at will for clarity). Each Learning Action can be submitted to editing, which opens up a LA Editor in a new tab. More details on LA Editor can be found in ***D3.6 – Experience Engine M24 [2].***

The user can quickly create new, empty Learning Actions with the Create functionality available through a right click in the working zone. An empty Learning Action will need further editing before it can be published on the MaTHiSiS cloud, but letting the user completely define his SLA in just one time without having to constantly swap to the LA Editor was an important point considered in this design decision. Once the SLA is completed, even with empty Learning Actions, the user can publish his SLA to the cloud with the Save to the cloud option available in the Toolbar.

Publishing an SLA to the MaTHiSiS cloud is done using the SLA OpenAPI method POST at the address api/sla/postSLA. This method is described in the SLA OpenAPI implementation details, present in this document at section 10.

The Browse panel on the left allows the user to view and access the Learning Actions stored on the MaTHiSiS cloud, and to add existing Learning Actions (made by him/her or by another MaTHiSiS user) to his/her SLA, exploiting the reusability concept.

Learning Actions are retrieved in the Browse panel using the LA Open API method GET at the adress /api/LA/LearningActions. This method is described in details in the ***D3.6 - Experience Engine M24[2]***.

Apart from saving to and downloading from the Cloud, the toolbar offers an option to save the SLA under edit to the disk as local copy, which can be useful in 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.

3.3.1.2 LG Editor

The Learning Content Editor also features a LG Editor tool, described in detail in ***D3.4 - The MaTHiSiS Learning Graphs M24[10]***. The aim of this tool is to build Learning Graphs with SLAs that can be added as nodes, and connected to Learning Goals with weighted edges describing the SLAs importance towards the Learning Goals.

In order to fulfill the goal of the LCE, the following functionalities have been put into place in the LG Editor:

- Create new, SLAs placeholders directly in the LG Editor without having to create them beforehand in a SLA Editor. This allows the user to stay focused on the graph s/he is currently editing while still being able to create all the content s/he needs.
- Open a SLA Editor to edit an existing SLA in a Learning Graph. This ensures that the navigation between the different editor tools is quick and easy, while the tools still remain independent from each other.
- Have at disposal a list of existing SLAs that can be reused in newer Learning Graphs.

4. Conclusion

This document describes the concept of the Smart Learning Atom (SLA), which is central to the approach taken in the MaTHiSiS project. It allows the description of an atomic and independent piece of knowledge or skill. This atomic and independent piece of knowledge will allow the users of the MaTHiSiS system to re-use their work in other similarly structured learning scenarios, as is often the case in modern schools or work environments.

In addition, this document gives details on the implementation of this concept in the second release of the MaTHiSiS platform. The implementation has evolved during the first two years of project, based on the feedback of end users gathered the during Driver and Assisted Pilot phases. However, implementation based on the feedback took the initial architecture into account and the approach taken has remained the same. The front-end has, as would be expected, evolved throughout the project, following the end-users requirements, but also with the additional insight concerning concrete uses in classes or during professional training with real people involved in education and learning during the two Pilots phases (“driver” and “assisted”).

5. References

[1] Nottingham Trent University (ed.): D.2.2 *Full Scenarios for all Use Cases*. Deliverable of the MATHISIS project, 2016.
[2] DIGINEXT (ed.): D3.6 Experience Engine M24. Deliverable of the MaTHiSiS project, 2017.
[3] ATOS (ed.): D7.1 *Integration Strategy and planning*. Deliverable of the MaTHiSiS project, 2016.
[4] DIGINEXT (ed.): D7.2 *MaTHiSiS platform, 1st release*. Deliverable of the MaTHiSiS project, 2017.
[5] ATOS (ed.): *D7.3 MaTHiSiS platform, 2nd release*. Deliverable of the MaTHiSiS project, 2017.
[6] MaTHiSiS Description of Action, 2016.
[7] UM (ed.): *D6.2 The MaTHiSiS Learning Graph Engine M24*. Deliverable of the MaTHiSiS project, 2017.
[8] UM (ed.): *D6.4 Synchronous and Asynchronous collaboration among platform agents M24*. Deliverable of the MaTHiSiS project, 2017.
[9] DIGINEXT (ed.): *D3.9 MaTHiSiS Frontend Components M24*. Deliverable of the MaTHiSiS project, 2017.
[10] Centre For Research and Technology Hellas (ed.): *D3.4 The MaTHiSiS Learning Graphs M24*. Deliverable of the MaTHiSiS project, 2017.

6. Appendix I: SLA Data Structures

The structures of the unpersonalised SLAs, personalised SLA instances and runtime 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. They are also presented, in context with the rest of the platform's structures, in Deliverable D7.3. As explained in Section 2, the unpersonalised SLAs reside on the Learning Content Space (LCS) and the personalised instances (hereafter referred to SLAIs) on the User Space (US). Fields marked with (*) constitute obligatory components of the structures.

lcsSmartLearningAtom			
Key	**Description**	**Value**	**Related to**
_id*	The unique identifier of the SLA	ObjectId	-
SLA_NAME*	The label of the SLA	String	-
SLA_DESCR	Details about what this SLA is about	String	-
CREATOR_ID*	The unique identifier of the user (tutor) that created this SLA	ObjectId	Collection *users*: _id Only for "role" : "Tutor"
CREATED	The date and time when this SLA was first created	ISODate	-
LAST_ MODIFIED	The date and time when this SLA was last modified	ISODate	-
LEARNING_ ACTIONS*	The list of PA-agnostic, generic LAs that are attached to (actuate) this SLA. There should be at least one LA attached to each SLA. This list is just a pointer to the full structure of each Learning Action.	Array	-
LA_NAME*	The unique name of the Learning Action	String	-
_id*	(Within the LA list:) the unique DB identifier of the Learning Action	ObjectId	(Learning Action) _id
RELATION_DE GREE	(For future releases). A weight in [0, 1] that denotes the relevance of the LA to the particular SLA. Default: 1.0 – in the first release	String	-

Table 4: Unpersonalised SLA data structure (Collection lcsSmartLearningAtom)

usSmartLearningAtomInstance			
Key	**Description**	**Value Type**	**Related to**
_id*	The unique DB identifier of the SLA instance	ObjectId	(US_SLAI_rtm) SLAI_id

SLA_NAME*	The unique name of the SLA	String	-
SLA_DESCR	Details about the SLA	String	-
SLA_ID*	The unique identifier of the corresponding unpersonalised SLA	ObjectId	(LCS_SLA) _id
LEARNER_ID*	The unique identifier of the user (learner) that this SLAI belongs to	ObjectId	(Users) _id, for role: learner
SLAI_WEIGHT	A weight in [0, 1] that denotes the achievement level of the particular learner concerning this knowledge/skill/ competence (SLA). Initial (no previous info available): 0.3	String	-
CREATED	The date and time when this SLA was first created	ISODate	-
LAST_ MODIFIED	The date and time when this SLA was last modified	ISODate	-

Table 5: Personalised SLAI data structure (US_SmartLearningAtomInstance)

usSmartLearningAtomInstance_rtm			
Key	**Description**	**Value Type**	**Related to**
_id*	The unique DB identifier of the SLA instance	ObjectId	-
SLA_NAME*	The unique name of the SLA	String	-
SLA_DESCR	Details about SLA	String	-
TYPE	Type of runtime instance	String	-
SLA_ID*	The unique identifier of the corresponding unpersonalised SLA	ObjectId	(LCS_SLA) _id
SLAI_ID	The unique identifier of the corresponding (long-term) personal SLAI	ObjectId	(US_SLAI_rtm) SLAI_id
LEARNER_ID*	The unique identifier of the user (learner) that this SLAI belongs to	ObjectId	(Users) _id, for role: learner
SLAI_WEIGHT	A weight in [0, 1] that denotes the achievement level of the particular learner concerning this knowledge/skill/ competence (SLA). Initial (no previous info available): 0.3	String	-
SESSION_ID	The unique identifier of the learning session when this SLAI's weight was modified during runtime	ObjectId	Collection *LearningSessions*: _id,

CREATED	The date and time when this SLAI was first created	ISODate	-
LAST_ MODIFIED	The date and time when this runtime SLAI was last modified. Always same as CREATED, maintain only for uniformity	ISODate	-

Table 6: Personalised SLAI runtime record data structure (usSmartLearningAtomInstance_rtm)

7. Appendix I: SLA lib Open API documentation

The tables below detail the functionalities of the REST calls available through the SLA 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/sla/getSLAs**		
Method	GET		
Content type	Application/JSON		
Description	Return the list of all SLAs 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**
	-	-	{ "slas": [{ "sla_id": {sla_id}, "sla_name": {sla_name}, }, ...] }

Table 7: SLA Open API - GET api/sla/getSLAs

URL pattern	**GET api/sla/getSLA**		
Method	GET		
Content type	Application/JSON		
Description	Return an unpersonalised SLA for a given id (and optionally, name as cross-reference)		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no SLA ID is provided.		
	HTTP 404 status code if no SLA 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={sla_id } & label={sla_name}	{ "_id": {SLA_id}, "SLA_DESCR" : "{SLA description}", "SLA_NAME" : "{SLA_name}", "LEARNING_ACTIONS" : [{ "LA_NAME" : "{LA_name}", "_id" : {LA_id}, "RELATION_DEGREE" : "1.0" }, ….], "LAST_MODIFIED" : {isodate} "CREATED" : {isodate} "CREATOR_ID" : {tutor_id}) }

Table 8: SLA Open API - GET api/sla/getSLA

URL pattern	**GET api/sla/getLAs**		
Method	GET		
Content type	Application/JSON		
Description	Return the list of learning actions attached to the specific unpersonalised SLA		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no SLA ID is provided.		
	HTTP 404 status code if no SLA 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={sla_id }	{ "LEARNING_ACTIONS" : [{ "LA_NAME" : "{LA_name}", "_id" : {LA_id}, "RELATION_DEGREE" : "1.0" }, ….], }

Table 9: SLA Open API - GET api/sla/getLAs

URL pattern	**GET api/sla/getSLAIs**
Method	GET
Content type	Application/JSON

Description	Return the list of all personal SLA instances in the MaTHiSiS repository. If a learner id is defined, return the list of all SLAIs that this learner works on. If a SLA id is defined, return the list of all SLAIs that are instantiated from this SLA.		
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 slaid	?uid={learner_id} &slaid={sla_id}	{ "learner_id" : {learner_id}, "slas": [{ "slai_id": {slai_id}, "sla_id": {sla_id}, "sla_name": {sla_name}, }, ...] }

Table 10: SLA Open API - GET api/sla/getSLAIs

URL pattern	**GET api/sla/getSLAI**		
Method	GET		
Content type	Application/JSON		
Description	Return a personalised SLAI for a particular learner, given the SLAI id (and optionally, name as cross-reference)		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no SLAI ID is provided.		
	HTTP 404 status code if no SLAI 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={slai_id }	{ "_id" : {SLAI_id}, "LEARNER_ID" : {LEARNER_id}, "SLA_ID" : {SLA_id}, "SLAI_WEIGHT" : "{∈[0,1]}", "SLA_NAME" : "{SLA_name}", "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate } }

Table 11: SLA Open API - GET api/sla/getSLAI

URL pattern	**GET api/sla/getSLAIs/rtm**
Method	GET

Content type	Application/JSON		
Description	Return the list of all personal runtime SLA instances in the MaTHiSiS repository. If a learner id is defined, return the list of all runtime SLAIs that this learner has worked on. If a SLAI id is defined, return the list of all runtime SLAIs pertaining to this SLAI. If a session id is defined, return the list of all runtime SLAIs 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 slaiid sid	?uid={learner_id} &slaiid={slai_id} &sid={session_id}	{ "learner_id" : {learner_id}, "slas": [{ "session_id" : {session_id}, "slai_rtm_id": {slai_rtm_id}, "slai_id": {slai_id}, "sla_name": {sla_name}, }, ...] }

Table 12: SLA Open API - GET api/sla/getSLAIs/rtm

URL pattern	**GET api/sla/getSLAI/rtm**		
Method	GET		
Content type	Application/JSON		
Description	Return a personal runtime record of an SLAI for a particular learner, given the SLAI id (and optionally, name as cross-reference)		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no runtime SLAI ID is provided.		
	HTTP 404 status code if no runtime SLAI 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={slai_id }	{ "_id" : {SLAI_rtm_id}, "LEARNER_ID" : {LEARNER_id}, "SLAI_ID" : {SLAI_id}, "SLA_ID" : {SLA_id}, "SLAI_WEIGHT" : {s∈[0.0,1.0]}, "SLA_NAME" : "{SLA_name}", "SESSION_ID" : {SESSION_id}, "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate} }

Table 13: SLA Open API - GET api/sla/getSLAI/rtm

URL pattern	**POST api/sla/postSLA**		
Method	POST		
Content type	Application/JSON		
Description	Create an unpersonalised SLA on the MaTHiSiS DB under the lcsSmartLearningAtom collection. Automatically detects SLA id from input structure. Missing non-mandatory fields of the input model are automatically filled in with default/initial values.		
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 SLA 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":{SLA_id}, "SLA_DESCR" : "{SLA description}", *"SLA_NAME" : "{SLA_name}", *"LEARNING_ACTIONS" : [{ "LA_NAME" : "{LA_name}", "_id" : {LA_id}, "RELATION_DEGREE" : "1.0" }, ...], "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate}, *"CREATOR_ID" : {tutor_id} }

Table 14: SLA Open API - POST api/sla/postSLA

URL pattern	**POST api/sla/postSLAI**

Method	POST		
Content type	Application/JSON		
Description	Create a personalised SLA instance for a given learner on the MaTHiSiS DB under the usSmartLearningAtomInstance collection. Automatically detects SLAI id from input structure. Missing non-mandatory fields of the input model are filled in with default/initial values.		
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 SLAI 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" : {SLAI_id}, *"LEARNER_ID" : {LEARNER_id}, *"SLA_ID" : {SLA_id}, "SLAI_WEIGHT" : "{∈[0.0,1.0]}", *"SLA_NAME" : "{SLA_name}", "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate} }

Table 15: SLA Open API - POST api/sla/postSLAI

URL pattern	**POST api/sla/postSLAI/rtm**		
Method	POST		
Content type	Application/JSON		
Description	*This call is evoked solely from the SLA lib and LG lib.* Create a personalised runtime SLA instance record for a given learner and session on the MaTHiSiS DB under the usSmartLearningAtomInstance_rtm collection. Automatically detects SLAI id from input structure. Automatically produces the SLAI runtime id and inserts the session id to the serialisation of the SLAI_rtm on the MaTHiSiS DB. The 'type' parameter defines which process has updated the weight; accepted fields are CREATION or RESET or DSS_PERSONALIZATION or DSS_ADAPTATION or LGE_PERSONALIZATION or LGE_ADAPTATION.		
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
	sid type	?sid={session_id} &type="{CREATION/RESET/DSS_ PERSONALIZATION/DSS_ADAPTA TION/ LGE_PERSONALIZATION/LGE_AD APTATION}"	{ "_id" : {SLAI_id}, *"LEARNER_ID" : {LEARNER_id}, *"SESSION_ID" : {SESSION_id}, *"SLA_ID" : {SLA_id}, *"SLAI_WEIGHT" : "{ ∈[0.0,1.0]}", *"SLA_NAME" : "{SLA_name}", "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate} }

Table 16: SLA Open API - POST api/sla/postSLAI/rtm

URL pattern	**POST api/sla/updateSLAIweight**		
Method	POST		
Content type	Application/JSON		
Description	Update the weight of a given personalised SLA instance for a given learner without having to repost the entire SLAI structure. Not applicable to runtime SLAIs. The 'type' parameter defines which process has updated the weight; accepted fields are CREATION or RESET or DSS_PERSONALIZATION or DSS_ADAPTATION or LGE_PERSONALIZATION or LGE_ADAPTATION		
Responses	HTTP 200 status code if successful		
	HTTP 400 status code if no SLAI ID is provided OR no session id is provided.		
Parameters	**Name**	**URL pattern**	**Input Model**
	id* sid* uid type	?id={slai_id} &sid={session_id} &uid={learner_id} &type="{CREATION/RESET/DSS_ PERSONALIZATION/DSS_ADAPTA TION/ LGE_PERSONALIZATION/LGE_AD APTATION}"	*A String representation of a double ∈[0.0,1.0]*

Table 17: SLA Open API - POST api/sla/updateSLAIweight

URL pattern	**PUT api/sla/putSLA**
Method	PUT
Content type	Application/JSON
Description	Create or update an unpersonalised SLA on the MaTHiSiS DB under the lcsSmartLearningAtom collection. Automatically detects SLA id from input structure. If the SLA with the provided id doesn't already exist in the DB it creates a new entry,

	otherwise it updates the fields of the existing retrieved entry. Missing non-mandatory fields of the input model are automatically filled in with default/initial values.		
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": {SLA_id}, "SLA_DESCR" : "{SLA description}", *"SLA_NAME" : "{SLA_name}", *"LEARNING_ACTIONS" : [{ "LA_NAME" : "{LA_name}", "_id" : {LA_id}, "RELATION_DEGREE" : "1.0" }, ….], "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate}, *"CREATOR_ID" : {tutor_id} }

Table 18: SLA Open API - PUT api/sla/putSLA

URL pattern	**PUT api/sla/putSLAI**		
Method	PUT		
Content type	Application/JSON		
Description	Create a personalised SLA instance for a given learner on the MaTHiSiS DB under the usSmartLearningAtomInstance collection. Automatically detects SLAI id from input structure. If the SLAI with the provided id doesn't already exist in the DB it creates a new entry, otherwise it updates the fields of the existing retrieved entry. Missing non-mandatory fields of the input model are filled in with default/initial values.		
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" : {SLAI_id}, *"LEARNER_ID" : {LEARNER_id}, *"SLA_ID" : {SLA_id}, "SLAI_WEIGHT" : "{∈[0.0,1.0]}", *"SLA_NAME" : "{SLA_name}", "LAST_MODIFIED" : {isodate}, "CREATED" : {isodate} }

Table 19: SLA Open API - PUT api/sla/putSLAI

URL pattern	**DELETE api/sla/deleteSLAs**	
Method	DELETE	
Description	Delete (drop) all SLAs under the lcsSmartLearningAtom collection in the MaTHiSiS DB	
Responses	HTTP 200 status code if successful	
Parameters	**Name**	**URL pattern**
	-	-

Table 20: SLA Open API - DELETE api/sla/deleteSLAs

URL pattern	**DELETE api/sla/deleteSLAIs**	
Method	DELETE	
Description	Delete (drop) all SLAIs under the usSmartLearningAtomInstance collection in the MaTHiSiS DB. If a learner id is defined, delete only the SLAIs that this learner has worked on. If a SLA id is defined, delete only the SLAIs instantiating this SLA.	
Responses	HTTP 200 status code if successful	
Parameters	**Name**	**URL pattern**
	uid slaid	?uid={learner_id} &slaid={sla_id}

Table 21: SLA Open API - DELETE api/sla/deleteSLAIs

URL pattern	**DELETE api/sla/deleteSLAIs/rtm**
Method	DELETE
Description	Delete (drop) all runtime SLAIs under the usSmartLearningAtomInstance_rtm collection in the MaTHiSiS DB. If a learner id is defined, delete only the SLAIs that this learner has worked on. If a SLAI id is defined, delete only the runtime SLAIs pertaining to this SLAI. If a session id is defined, delete only the runtime SLAIs created during that session.
Responses	HTTP 200 status code if successful

Parameters	Name	URL pattern
	uid slaid sid	?uid={learner_id} &slaid={slaii_id} &sid={session_id}

Table 22: SLA Open API - DELETE api/sla/deleteSLAIs/rtm

URL pattern	DELETE api/sla/deleteSLA	
Method	DELETE	
Description	Delete (drop) a specific SLA in the lcsSmartLearningAtom 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={sla_id}

Table 23: SLA Open API - DELETE api/sla/deleteSLA

URL pattern	DELETE api/sla/deleteSLAI	
Method	DELETE	
Description	Delete (drop) a specific SLAI in the usSmartLearningAtomInstance collection in the MaTHiSiS DB.	
Responses	HTTP 200 status code if successful	
	HTTP 400 status code if no SLAI ID was provided	
Parameters	Name	URL pattern
	id	?id={slai_id}

Table 24: SLA Open API - DELETE api/sla/deleteSLAI

URL pattern	DELETE api/sla/deleteSLAI/rtm	
Method	DELETE	
Description	Delete (drop) a specific runtime SLAIs in the usSmartLearningAtomInstance_rtm collection in the MaTHiSiS DB.	
Responses	HTTP 200 status code if successful	
	HTTP 400 status code if no runtime SLAI ID was provided	
Parameters	Name	URL pattern

	id	?id={slai_rtm_id}

Table 25: SLA Open API - DELETE api/sla/deleteSLAI/rtm