SEVENTH FRAMEWORK PROGRAMME

SPLENDID
Personalised Guide for Eating and Activity Behaviour for the Prevention of Obesity and Eating Disorders

Seventh Framework Programme

CallFP7-ICT-2013-10

Project Acronym:	**SPLENDID**
Grant Agreement N^{o}:	**610746**
Project Type:	**COLLABORATIVE PROJECT: Small or medium scale focused research project (STREP)**
Project Full Title:	**Personalised Guide for Eating and Activity Behaviour for the Prevention of Obesity and Eating Disorders**

D3.2 Formal ontology for the eating and activity behaviour domain

Nature:	**P** (R: Report, P: Prototype, O: Other)
Dissemination Level:	**PU** (CO: Confidential, PU: Public)
Version #:	**1.0**
Date:	**30 September 2014**
WP number and Title:	**WP 3 Information Processing and Inference**
Deliverable Leader:	**AUTH**
Author(s)/Contributor(s):	Authors: **Ioannis Sarafis (AUTH), Christos Diou (AUTH) and Anastasios Delopoulos (AUTH)** Contributors:
Status:	**Submitted** (Draft, Peer-Reviewed, Submitted, Approved)

Document History

Version[1]	Issue Date	Status[2]	Content and changes
0.1	16.09.2014	Draft	First version
0.2	19.09.2014	Draft	Revised
1.0	30.09.2014	Submitted	Final version

Peer Review History[3]

Version	Peer Review Date	Reviewed By

[1] Please use a new number for each new version of the deliverable. Use "0.#" for Draft and Peer-Reviewed. "x.#" for Submitted and Approved", where x>=1.Add the date when this version was issued and list the items that have been added or changed.

[2] A deliverable can be in one of these stages: Draft, Peer-Reviewed, Submitted and Approved.

[3]Only for deliverables that have to be peer-reviewed

Table of contents

Abbreviations and Acronyms

List of Abbreviations and Acronyms

DOW	Description of Work
WP	Work Package
UC	Use Case
UI	User Interface
BNF	Backus Normal Form - BNF is a notation technique for context free grammars

Nomenclature

`(Health) Professional`: A doctor, a dietician, a therapist or a lifestyle coach.

`Adult`: A young adult person that uses SPLENDID as a lifestyle product.

`Student`: An adolescent that participates in the primary/secondary screening processes and the guidance `Events`.

`Subject`: A `Student` or `Adult` user.

`Assistant`: A user that helps the `Health Professional` to run some types of `Events`.

`Mandometer`©: A personal scale registering weight-loss data from a plate, used to take measurements of the meals.

`Activity Meter`: A portable device that logs the user's physical activity through the day.

`Chewing Sensor`: A wearable sensor that is used to identify snacking `Events`. Also it can be used as alternative of the `Mandometer`© scale to record meals.

`Smartphone`: A mobile phone with an operating system and internet capabilities.

`Event`: Refers to all type of screening and guidance processes featured by SPLENDID. An `Event` refers to a single `Student`/`Adult`. In our context these types of `Events` may last several days.

`Primary Screening Event (PSE)`: This is the type of `Event` that refers to the primary screening process at schools.

`Primary Screening Stage (PSS)`: Refers to the phase of collecting data of `Primary Screening Events` from a school population during a period of time (typically some days).

`Behavioural Assessment Event (BAE)`: This is the type of `Event` that refers to the secondary screening process for the `Students` and the behavioural assessment process for the `Adults`.

`Personalized Guidance Event (PGE)`: This is the type of `Event` that refers to the personalized guidance process for the `Students` and the `Adults`.

`Primary Screening Application`: A `Smartphone Application` for the `Primary Screening Events`.

`Smartphone Application`: An application for the `Behavioural Assessment Events` and the `Personalized Guidance Events` used by the `Students` and the `Adults`.

Mobile Application: The interface of the SPLENDID system that is accessed through the Smartphone. This encompasses the Primary Screening Application and the Smartphone Application.

Web Interface: An interface accessed through a web browser. It is used for the management of the SPLENDID system by the Health Professionals, as well as the Adults or the Students, mainly during the Personalized Guidance Events.

Executive Summary

This deliverable documents the initial outcomes of Task 3.2 of the SPLENDID project (entitled "Goal semantics expression and monitoring"). More specifically, it focuses on i) formally expressing the entities involved in the various phases of SPLENDID operation via a prototype ontology for the eating and activity behaviour domain ii) defining a formal language for expressing goals to be used during Personalised Guidance Events (cf deliverable D1.1 and D1.2 for details) and iii) documenting the use of a prototype software tool for parsing and validating goals expressed in the goal language.

AUTH developed the SPLENDID ontology based on the Use Case analysis of Deliverable D1.2 ("Use Cases Specifications") and the Functional Requirements of Deliverable D1.1 ("User Requirements"). These deliverables provided the necessary groundwork for the identification of the ontology classes, their properties and the relations between them. Each class and property of the ontology is thoroughly documented, both in this document as well as via built-in annotations in the prototype. In addition, this document also provides detailed examples on how to define data/storage models and communication interfaces based on the ontology.

The prescription of personalised goals can be achieved using the ontology in combination with a formal language for goal expression. AUTH has developed a UI tool for evaluation and parsing of the prescribed goals; documentation and usage examples are presented in this document.

The outcomes of this deliverable will be used in subsequent phases of SPLENDID development, such as i) the use of the ontology as a reference for designing the SPLENDID data model and databases ii) the use of the ontology for defining the communication interfaces between the different local and web service components iii) the use of the formal goal language for storing goals expressed by health professionals in the SPLENDID database and iv) the use of the UI tool for defining a range of different goal types during SPLENDID development.

The prototypes presented in this document allow the development of the goal monitoring modules, to be presented in deliverable D3.3 ("Preliminary version of risk assessment and goal monitoring"), which will assess the proximity of an individual subject to goals expressed in the formal goal language.

1 Introduction

1.1 Purpose of the document

Deliverable D3.2 entitled "Formal Ontology for the Eating and Activity Behaviour Domain" is the first deliverable of the Work Package 3 (WP3 - Information Processing and Inference), presenting the work done until Month 12 of the project in the context of Task 3.2 ("Goal semantics expression and monitoring"). The present deliverable is based on Deliverable D1.1 ("User Requirements") and mostly, on Deliverable D1.2 ("User Cases Specifications").

The scope of this document is:

- The development of a formal ontology for the eating and activity behaviour domain.
- The definition of a formal language that will allow SPLENDID developers to express, store and interpret different personalised goals prescribed by the Health Professionals via the SPLENDID professional web application.
- The development of a User Interface (UI) that will allow definition and validation of specific eating and activity goals, expressed in this formal language.

1.2 Methodology

The first goal of this deliverable was the development of the ontology. The Use Cases (UCs) of D1.2 were studied to ensure complete coverage of the UCs by the classes of the ontology and their underlying relations. The ontology was designed with three objectives:

- To allow the description of the `Subject`'s behaviour in the eating and activity domains.
- To specify the data model. The ontology clearly presents all relevant entities and their relations and dictates the necessary entities for the design of the data/storage model.
- To enable definition of behavioural goals, in combination with a formal goal expression language.

D1.1 and D1.2 identified three phases of the SPLENDID system. These are the *Primary Screening* phase, the *Behavioural Assessment* phase, and the *Personalised Guidance* phase. The UCs of each phase were further categorized and analysed at the application level, leading to UCs for the `Web Interface` and UCs for the `Mobile Application`. Following separation of the UCs into phases, the ontology classes and properties were first defined for each phase separately and were then combined to build the final ontology.

Next, based on the UCs of the *Personalised Guidance* phase and the constructed ontology, a formal language for the behavioural goal definition is given. The language is based on the ontology classes, their data properties and object properties. A variety of logical and comparison operators as well as the usage of special functions were introduced in the language to allow the definition of arbitrary goals and provide the flexibility for the definition of personalised behavioural goals tailored for each `Subject`.

Finally, a software tool was created that allows the manipulation and validation of the personalized goals. This tool allows goal parsing described using the introduced goal language, evaluates the syntax, and produces the parse tree.

1.3 Intended audience and reading suggestions

The SPLENDID ontology, the language for goal prescription, and the software tool will be combined with the modules of D3.3 (“Preliminary version of risk assessment and goal monitoring module”) into D3.4 (“Decision Support System”) of Work Package 3. Furthermore, the development teams of SPLENDID can rely on the ontology for the definition of the storage model and the communication interfaces. To this end, examples for using the ontology for creating a relational database schema and defining JSON objects [1] for communication interface are presented in the document.

The reader can read the sections sequentially, or can use the document as a reference for each individual class/property of the ontology.

1.4 Writing Conventions

The notions that are presented in the Nomenclature are given with `Courier New` in the document. Text that illustrates code and programs’ verbatim input/output is given with `Courier New` as well. Ontology classes, data properties, object properties, and database table names are *italicized*.

1.5 Overview of the document

Section 2 provides an overview of the UCs and documents in detail the ontology classes and their properties for each phase. In addition, examples for using the ontology for the design of the communication interface and the storage model are given. Section 3 describes the language for the definition of personalised goals and demonstrates the usage of the goal expression language tool whereas Section 4 presents the conclusions of the document. Finally, Annex A provides the formal goal language in BNF form and the graph of the implied automaton.

2 Ontology

This section presents the ontology for eating and activity domain. It is organised as follows: Section 2.1 presents an overview of the UCs and Section 2.2 presents an overview the ontology entities and relations identified from the UCs. Then, Sections 2.4 and 2.5 present examples on exploiting the ontology for the communication interface and storage model design.

2.1 Overview of UCs

Two categories of UCs were identified in D1.2 ("Use Cases Specifications"): the `Web Interface` UCs and, the `Mobile Application` UCs.

According to these UCs, the `Health Professionals` are the primary users of the `Web Interface`. Through the `Web Interface` they are able to:

- Register and manage `Student/Adult` and `Assistant` user accounts.
- Create and/or manage `Personalised Screening Stages` (`PSS`), `Primary Screening Events` (`PSE`), `Behavioural Assessment Events` (`BAE`), and `Personalised Guidance Events` (`PGE`).
- Assign `Student/Adult` and `Assistant` users to the `Events`.
- Give permissions to other `Health Professionals` for the `Events` they own.
- View and export the data and the extracted features collected from the sensors as a part of an `Event`.
- View the automatically risk and the risk explanation assigned by the system for the `Subjects` participating in a `PSE` or a `BAE`. The `Health Professionals` are able to finalise the risk by confirming or modifying the automatic risk.
- Prescribe behavioural goals for `Subjects` participating in a `PGE`. Furthermore, the `Health Professionals` are able to view the progress of the `Student/Adult` relatively to the goals.
- Assign appointments for the `Subjects` participating in a `PSE` or a `BAE`.

The `Student/Adults` participating in a `PGE` may be provided limited access to the `Web Interface`. The provided information via the `Web Interface` is a subset of the `Professional`'s web application and is limited in viewing part of the collected data, the prescribed goals and their improvement.

The `Mobile Application` UCs describe the provided functionality to the `Student/Adult` user via the `Smartphone Applications`. The UCs describe functionalities for:

- Recording meal using the `Mandometer`© scale and/or the `Chewing sensor`. The recorded meals can be part of a `PSE`, a `BAE` or a `PGE`.
- Recording the physical activity of the `Student/Adult` using the `Activity Sensor` during `BAE` and `PGE`.
- Answer a variety of questionnaires. The questionnaire can be related to an `Event` and be answered in a daily basis (e.g. end-of-day questionnaire, sleep/wake-up time questionnaires), during a registered meal or after a detected snacking event. During the `PGE`, the `Student/Adult` receives additional end-of-day reports which present summarized information with respect to the behavioural goals.

- The `Student/Adult` users participating in a `PGE` are able to view the prescribed goals and the overall proximity to them and receive personalised guidance via the `Mobile Application`.

The UCs also describe functionalities for the `Professional` and `Assistant` users that is provided through the `Mobile Application`. However, this is fairly limited and related to data extraction (by the `Professional`) and `PSE` preparation (by the `Assistant`).

2.2 Ontology Entities and Relations

Ontologies provide a formal way to represent knowledge and the goal of the SPLENDID domain-specific ontology is to formally describe the entities identified in the SPLENDID system, their attributes and relations between them. The ontology must accurately represent the data model of the system and describe the eating and activity behaviour domain which will allow the formal definition of personalised behavioural goals.

Common components of the ontologies are:

- *Classes*: Classes are used to classify individuals. Classes represent concepts, type of objects or things in an abstract way.
- *Subclasses*: Classes that can be subsumed by other class. The parent class inherits its properties to the children classes. Classes and subclasses define the class hierarchy of the ontology.
- *Individuals*: Instances of the classes that may refer to specific objects or concepts. Although the designed ontology provides the means for classifying individuals, class instances were not included in the ontology.
- *Attributes* or *data properties*: Properties and parameters of the classes and individuals.
- *Relations* or *object properties*: Describe the relations between the classes.

Figure 1 displays the identified ontology hierarchy; the classes and their subclasses, where available. Next, Sections 2.2.1, 2.2.2 and 2.2.3 and describe in depth the classes, their data and object properties for *Primary Screening*, *Behavioural Assessment* and *Personalised Guidance* phase respectively.

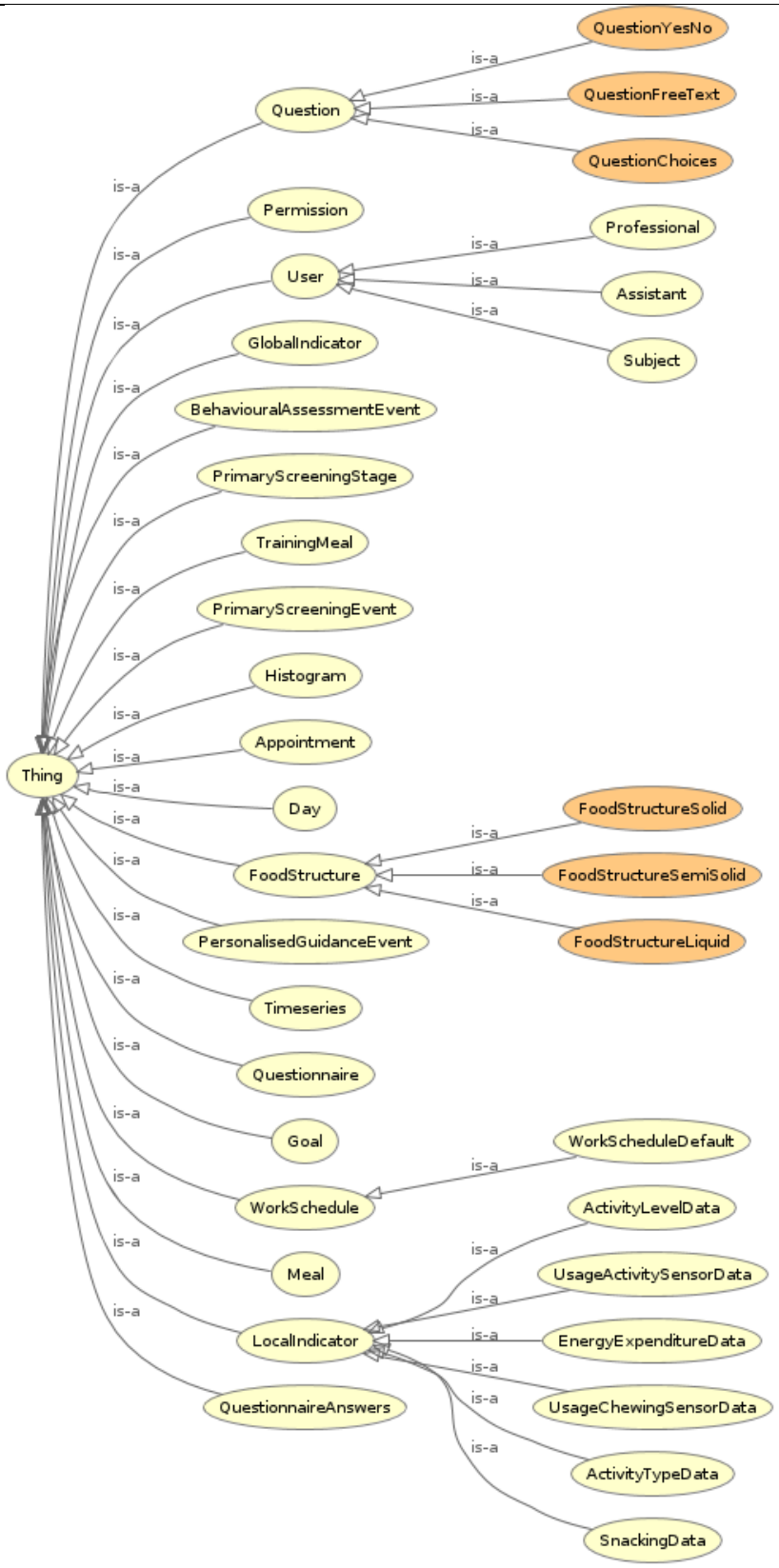

Figure 1 Overview of the ontology's class hierarchy.

2.2.1 Primary Screening Phase

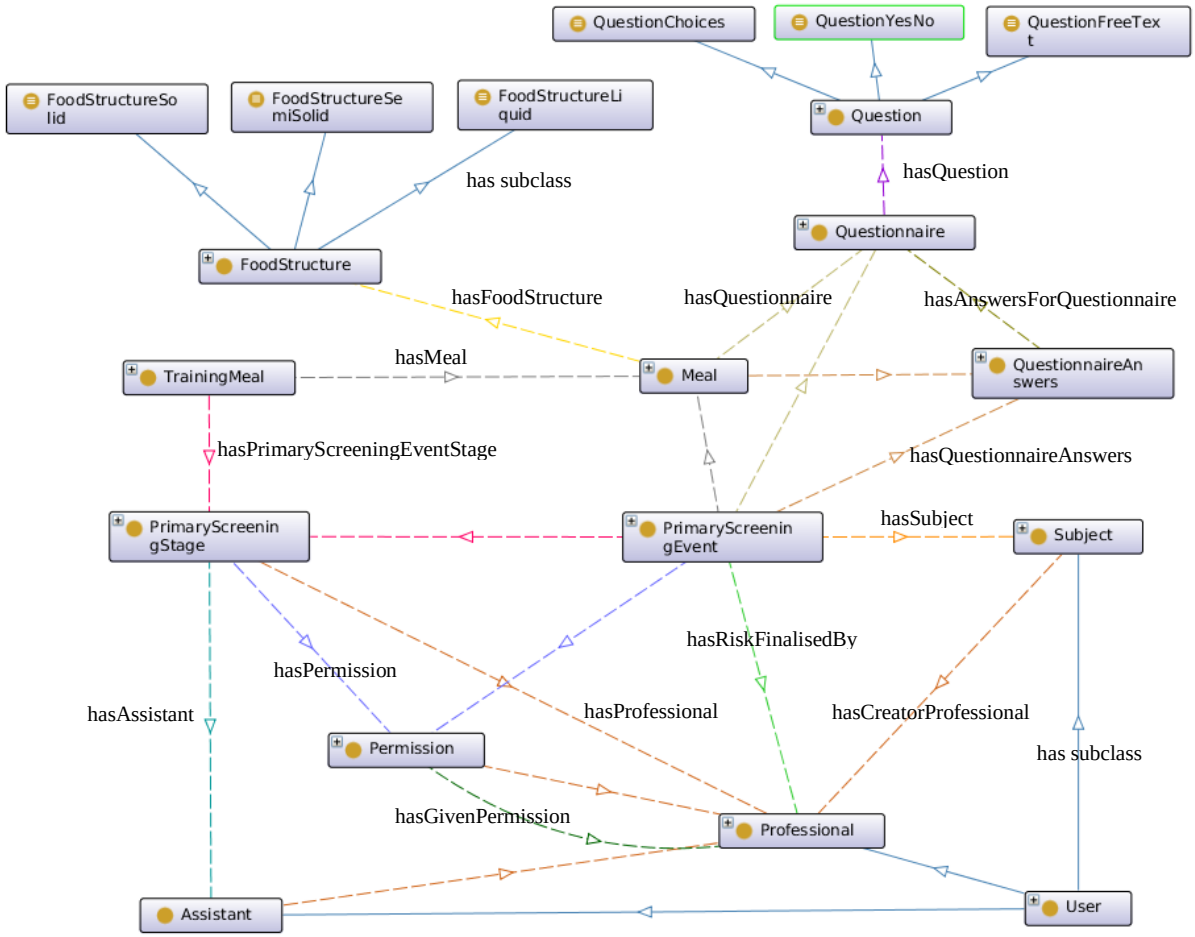

Figure 2 Ontology classes and their relations for Primary Screening Phase. Each property is identified by a colour/line type combination and is only mentioned once.

Figure 2 displays the classes and the object properties for the classes identified from the UCs of the *Primary Screening* Phase. Table 1 displays the classes, the classes' description and their data properties. Due to the overlap of the object properties among the phases, they are described all together in Section 2.2.4.

Table 1 Primary Screening Phase Classes and Data Properties

Class Name	***Class Description***	***Data Properties***
User	The users of the system. Three types of user are defined as subclasses: `Assistant`, `Professional`, and `Subject`.	• *userUsername:* The username of the user credentials. • *userPassword:* The password of the user credentials. • *description:* Free text description added by `Professionals` for `Assistant` and `Subject` users.

Assistant	Class for the `Assistant` users.	• *assistantID:* Unique identifier for `Assistant` users. • *userContactDetails:* Contact details. • *userEmail:* Email address. • *userFirstName:* First name. • *userLastName:* Last name.
Professional	Class for the `Professional` users.	• *professionalID:* Unique identifier for `Professional` users. • *userContactDetails:* Contact details. • *userEmail:* Email address. • *userFirstName:* First name. • *userLastName:* Last name.
Subject	Class for the `Student`/`Adult` users. They are described using the same class since the system's functionality (as described in UCs) is almost the same for the `Students` and the `Adults` with minor changes. Data property *subjectType* is used to discrete the type. The Student/Adult storage entries must be anonymised in SPLENDID system, hence, no personal identification information was added as data property.	• *subjectID:* Unique identifier for `Subject` users. • *subjectBirthYear:* Birth year in order to calculate the `Subject`'s age. Only the year of the birth date may be stored in the system, since the exact birth date can be considered as personal identification information. • *subjectGender:* `Subject`'s gender. • *subjectHandedness:* Handedness of the `Subject` (left/right-handed). • *subjectHeight:* `Subject`'s height. • *subjectWeight:* `Subject`'s weight. • *subjectType:* Whether the `Subject` is a `Student` or `Adult` user. • *riskEatingDisorder:* `Subject`'s risk on developing eating disorders. • *riskObesity:* `Subject`'s risk on developing obesity.
Questionnaire	This class was introduced to cover the need of	• *questionnaireType:* The type of the questionnaire, for example:

	questionnaires as indicated by a number of UCs. The questionnaires can be related to: • The questions related with a meal. • The general questions asked to the students during PSE. • Wake-up/end-of-day questionnaires and popup questionnaires during BAE and PGE.	○ BAE_ActivityDetectionPopup ○ BAE_Meal, ○ BAE_SnackDetectionPopup, ○ PGE_ActivityDetectionPopup ○ PGE_Meal, ○ PGE_SnackDetectionPopup, ○ PSE_AfterMeal, ○ PSE_BeforeMeal, ○ PSE_Meal
Question	The class that represents the questions of the questionnaires. It was separated from questionnaire to allow better description of the question types via the definition of subclasses.	• *questionType:* The type of the question. For example: ○ *Multiple choises,* ○ *Free text,* ○ *Yes/no questions.*
QuestionYesNo	Subclass for the yes/no questions.	
QuestionChoices	Subclass for the multiple choices questions.	• *answerChoices:* The available answers to choose from.
QuestionFreeText	Subclass for the questions with free text answers.	
QuestionnaireAnswers	The answers to a questionnaire. The class was introduced to indicate that the storage of the answers is decoupled from the questions.	• *answers:* The answers to the questionnaire. A JSON object may be used for the storage of the answers. • *answersTimestamp:* The time the questionnaire was answered since a questionnaire may be answered multiple times during an Event.
Meal	Class that covers all types of meals (registered/ detected, planned/ snacking). The class contains the meal related data and indicators.	• *averageBiteFrequency*: Average bite frequency (bites/sec). • *averageChewingRate*: Average chewing rate (chews/sec). • *averageFoodIntake*: Average food intake (g/sec). • *biteSizeAcceleration:* Bite size

		acceleration (g/bite^2). • *biteSizeRate:* Bite size rate (g/bite). • *chewingRateAcceleration:* Chewing rate acceleration (chews/sec^2). • *estimatedKcal:* Estimated total caloric intake (Kcal). • *hasChewingSensor:* Boolean value; true if `Chewing Sensor` was used for the meal recording. • *hasMandometerSensor:* Boolean value; true if `Mandometer`© scale was used for the meal recording. • *initialBiteSize:* Initial bite size (g). • *initialChewingRate:* Initial chewing rate (chews/sec). • *initialFoodWeight:* The initial food weight of the meal. • *isConfirmedSnack:* Boolean value; true if the meal at hand is a snack confirmed by the Student/Adult. • *isRegisteredMeal:* Boolean value; true if the meal was registered by the `Subject`. • *mealCurveAlpha:* Food intake rate acceleration - parameter "a" of the meal curve (g/sec^2). • *mealCurveBeta:* Initial food intake rate - parameter "b" of the meal curve (g/sec). • *mealDetectionProbability:* If the meal was automatically detected, the value indicates the probability it was detected correctly. • *mealDuration:* Meal duration (sec). • *mealStartTime:* Timestamp for the meal start time. • *mealEndTime:* Timestamp for the meal end time. • *mealType:* The type of the meal. For example, the meal

		could be between: "Breakfast", "Dinner", "Lunch" and "Snack". • *numberChewsBeforeSwallow:* Average number of chews before swallowing during a meal. • *numberOfFoodAdditions:* Number of food additions during meal. • *pauseLengthBeforeEating:* The pause in seconds before eating. • *satietyFoodIntakeRatio:* Satiety to the total food intake (satiety/g). • *totalFoodIntake:* Total food intake (g). • *varianceOfBiteSize:* Bite size standard deviation (g). • *weightOfLeftOvers:* The weight of the food left on the plate when meal is over (g).
FoodStructure	The food structure of a meal. Because different food structure properties may be required depending on the food type three subclasses were introduced for solid, semi-solid and liquid food.	• *foodType:* The type of the food, e.g. solid, liquid, semi-solid.
FoodStructureLiquid	To define food structure properties when the food is identified as solid.	• *liquidType:* Type of the liquid, such as: ○ *Carbonated* ○ *Neutral* ○ *Other*
FoodStructureSemiSolid	To define food structure properties when the food is identified as semi solid.	
FoodStructureSolid	To define food structure properties when the food is identified as liquid.	• *isChewy:* Boolean value; true if the food is chewy. • *isCrispy:* Boolean value; true if the food is crispy. • *isWet:* Boolean value; true if the food is wet.

PrimaryScreeningEvent	The `Primary Screening Event` (`PSE`) for a `Student`.	• *automaticRiskCategory:* The risk category calculated automatically by the system. For example, the risk could fall into the following categories: "Eating Disorders Risk", "Obesity Risk" and, "Low Risk". • *automaticRiskExplanation:* A description on how the risk was calculated by the system. • *finalisedRiskCategory:* The finalised risk category for the `Event` assigned by the `Professional` and shall receive the same values with the *automaticRiskCategory* property. • *finalisedRiskComments:* Comments added by the `Professional` when the risk of the `Event` is finalised. • *riskEatingDisorder:* The automatically calculated value for the eating disorders risk. • *riskObesity:* The automatically calculated value for the obesity risk. • *isRiskCategoryFinalised:* Boolean value; true if the risk has been finalised by a Professional.
PrimaryScreeningStage	The class that groups `PSE` of the same primary screening process (e.g. same school and same age etc.). `PSE` of the same type are grouped in a `Primary Screening Stage` (`PSS`).	• *dateStart:* The start date of the `PSS`. • *dateEnd:* The end date of `PSS`. • *description:* A free text description for the `PSS` added by the `Professional`. • *location:* The location `PSS` takes place. • *plateWeight:* Since the weight of the plate may not be measured during the `PSE` meal recordings, the `Professional` shall be able to denote the weight of the plates of the school cafeteria directly into

		SPLENDID system.
TrainingMeal	The training meals during `PSS` that help students familiarise with the primary screening process. These meals are anonymous and there is no need to link them with the `Students`.	• *subjectGender:* The `Student`'s gender. • *foodImageUri:* If available, this property is the URI of the photo of the `Student`'s lunch.
Permission	Class to describe the permissions the owner `Professional` of an `Event` may grant to other `Professionals`.	• *permissionLevel:* A predefined permission level for the Event. For example: o *Full* o *Read Only*

2.2.2 Behavioural Assessment Phase

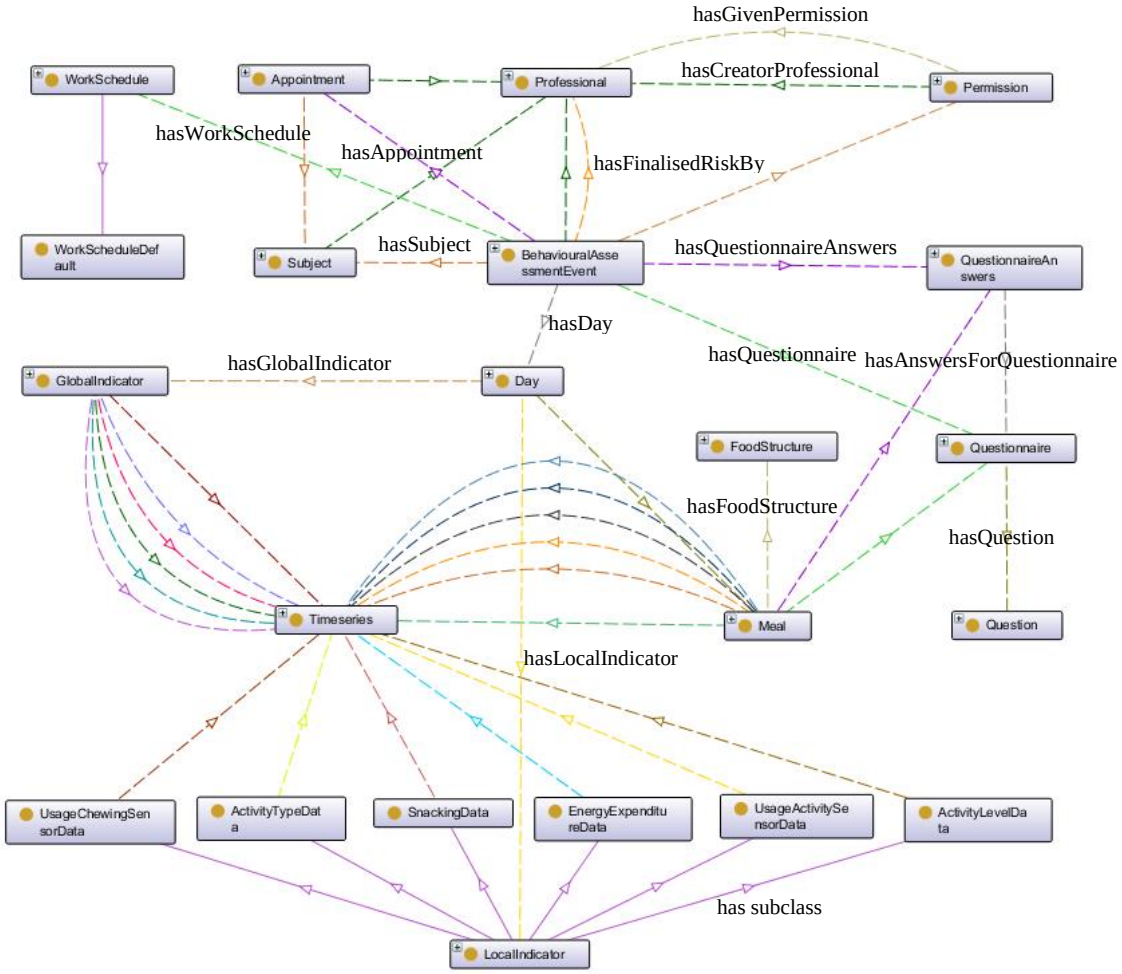

Figure 3 Ontology classes and their relations for Behavioural Assessment Phase. For clearanceness the object properties labels related to Timeseries class are omitted from the diagram (see Section 2.2.4 for the description of the object properties).

Figure 3 shows the classes and the class relations for the classes related to *Behavioural Assessment* phase and Table 2 displays the identified classes along with a description and their data properties.

Table 2 Behavioural Assessment Phase Classes and Data Properties

Class Name	***Class Description***	***Data Properties***
Professional	*Same as in Table 1*	
Subject		
Questionnaire		
Question		
QuestionnaireAnswers		
Meal		
FoodStructure		
Permission		
BehaviouralAssessmentEvent	The `Behavioural Assessment Event` (`BAE`) for a `Student`/`Adult`. A `BAE` refers to a single `Student`/`Adult` and typically lasts for two weeks.	• *automaticRiskCategory:* The risk category automatically calculated by the system. For example, the risk could fall into the following categories: "Eating Disorders Risk", "Obesity Risk" and, "Low Risk". • *automaticRiskExplanation:* A description on how the risk was calculated by the system. • *dateStart:* The start date of the `BAE`. • *dateEnd:* The end date of `BAE`. • *description:* A free text description for the `BAE` added by the `Professional`. • *finalisedRiskCategory:* The finalised risk category for the `Event` assigned by the `Professional`. The range of this value is the same as the one used for *automaticRiskCategory* property. • *finalisedRiskComments:*

		Comments added by the `Professional` when the `Event` risk is finalised. • *riskEatingDisorder:* The automatically calculated value for the eating disorders risk. • *riskObesity:* The automatically calculated value for the obesity risk. • *isRiskCategoryFinalised:* Boolean value; true if the risk has been finalised by a `Professional`.
Day	Class that represents a day of a `BAE` or a `PGE`. This class was introduced to allow flexibility in defining goals on a daily basis (see goal expression language of Section 3).	• *isWorkingDay:* Boolean value; true if it is a working day. Different types of goal may apply if the day is a working day or a day off. • *date:* The date (i.e. the date when the data were collected) for a `BAE` or `PGE`.
Appointment	The appointments are set by the `Professionals` for the `Subjects` during a `BAE` or `PGE`.	• *datetime:* Date and time of the appointment. • *description:* A free text description field. • *location:* The location the appointment will take place.
Timeseries	Wrapper class for time series data of the system.	• *jsonUri:* The URI of the JSON object that stores the time series. Section 2.3 proposes a JSON format for time series data.
GlobalIndicator	Global indicators extracted over a specified period of time, such as a day. Depending on the implementation, the global indicators could be calculated on-the-fly (e.g. for different requested periods) and not be stored in the database. Furthermore, some of the indicators may be needed only for presentation purposes and	• *averageIntakeRate:* Average intake rate across registered meals (g/sec). • *compliaceActivitySensor:* User compliance level for `Activity Sensor` (categorical and/or probability). • *complianceChewingSensor:* User compliance level for chewing sensor (categorical and/or probability). • *complianceMandometer:* User

	not be used in goal definition and/or automatic risk assignment.	compliance level for `Mandometer`© (categorical and/or probability). • *startTimestamp:* Start time of the period global indicators are calculated for. • *endTimestamp:* End time of the period global indicators are calculated for. • *estimatedDailyEnergyExpenditure:* Estimated daily energy expenditure during the period (average MET). • *estimatedTotalEnergyExpenditure:* Estimated total energy expenditure during the period (average MET). • *exerciseSessionDuration:* Exercise session duration (minutes). • *numberConfirmedSnackingEvents:* Number of confirmed snacking events for the requested period. • *numberDetectedSnackingEvents:* Number of detected snacking events by the sensors for the requested period. • *numberExerciseSessions:* Number of exercises sessions for the requested period. • *numberMeals:* Number of meals for the requested period. • *questionsAnsweredPercentage:* Percentage of the questions answered during the specified period.
LocalIndicator	Class for the local indicators collected during BAE and PGE. The subclasses of *LocalIndicator* represent the different types of data that may be collected/extracted from the `Chewing` and the `Activity Sensor`.	• *startTimestamp:* Start time of the period local indicators were collected. • *endTimestamp:* End time of the period local indicators were collected.

UsageChewingSensorData	Class containing time series for the usage (or the probability of usage) of the `Chewing Sensor` by the `Subject`.	
ActivityTypeData	Class containing time series for the `Subject`'s (calculated assumption of) activity type.	
SnackingData	Class containing time series for the probability a snacking event was detected by the sensors.	
EnergyExpenditureData	Class containing time series for the estimated energy expenditure.	
UsageActivitySensorData	Class containing time series for the usage (or the probability of usage) of the `Activity Sensor` by the `Subject`.	
ActivityLevelData	Class containing time series for the activity level extracted by the `Activity Sensor`.	
WorkSchedule	The working schedule (working days and days off for the `Subject`). The schedule may be defined using week days (e.g. Monday is workday, Sunday is day off) or by specific dates (for the subjects where the above format cannot be used).	• *scheduleDays:* Data property to indicate that the work schedule can be defined using days (e.g. Monday is working day, Sunday is day off, etc.). • *scheduleDates:* Data property to indicate that the work schedule can be defined using dates directly (e.g. 23-02-2015 is working day, 24-02-2015 is day off, etc.).
WorkScheduleDefault	Subclass of *WorkSchedule*; indicate that a default work schedule shall be defined in the system (e.g. Monday to Friday are working days, Saturday and Sunday are	

	days off).	

2.2.3 Personalised Guidance Phase

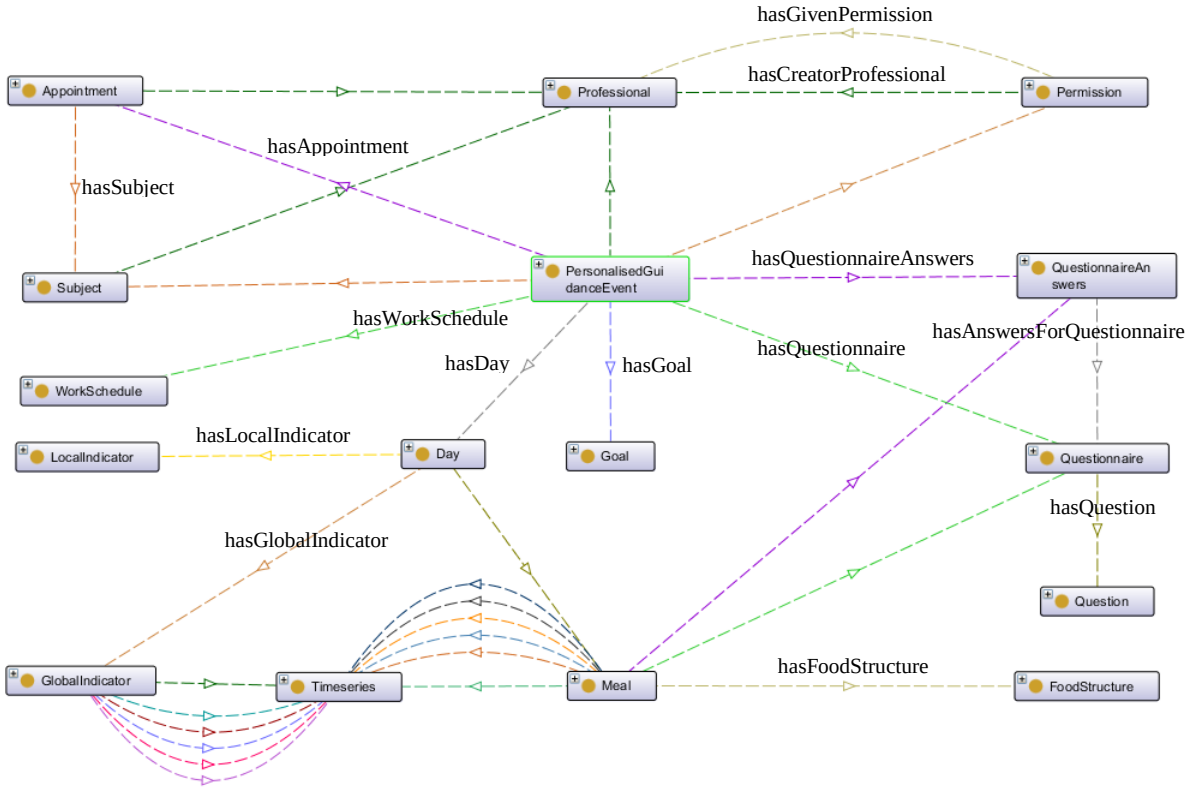

Figure 4 Ontology classes and their relations for Personalised Guidance Phase. For clearanceness the object properties labels of Timeseries class are omitted (see Section 2.2.4 for the description of the object properties).

Figure 4 shows the classes and the class relations for the classes related to *Personalised Guidance* phase and Table 3 displays the classes along with a description and their data properties.

Table 3 Personalised Guidance Phase Classes and Data Properties

Class Name	***Class Description***	***Data Properties***
Professional	*Same as in Table 1*	
Subject		
Questionnaire		
Question		
QuestionnaireAnswers		

Meal		
FoodStructure		
Permission		
Day	*Same as in Table 2*	
Appointment		
Timeseries		
GlobalIndicator		
LocalIndicator		
UsageChewingSensorData		
ActivityTypeData		
SnackingData		
EnergyExpenditureData		
UsageActivitySensorData		
ActivityLevelData		
WorkSchedule		
WorkScheduleDefault		
PersonalisedGuidanceEvent	The `Personalised Guidance Event` (`PGE`) for a `Student/Adult`. A `PGE` refers to a single `Student/Adult` and typically lasts for two weeks.	• *dateStart:* The start date of the `PGE`. • *dateEnd:* The end date of `PGE`. • *description:* A free text description for the `PGE` added by the `Professional`.
Goal	The behavioural goals assigned to `PGE` by the `Professionals`.	• *dateStart:* Start date of the period the *Goal* is evaluated/active. • *dateEnd:* Start date of the period the *Goal* is evaluated/active. • *goalLogicalExpression:* The logical expression that defines the goal. Section 3 describes the formal structure of the goal

		logical expressions. • *proximityToGoal:* Numerical or categorical value for the proximity to goal satisfaction. • *proximityToGoalExplanation:* If the goal is more complex, the explanation describes the contribution of each part for the proximity evaluation.

2.2.4 Object Properties

This section presents the object properties between the Ontology classes. The object properties define how individuals are related to other individuals in the form "DomainObject hasRelation RangeObject". Table 4 displays the object properties that are defined in the ontology.

Table 4 Object Properties

Object Property	***Domain/Range/Description***
hasActivityLevelHistogram	**Domain**: *GlobalIndicator* **Range**: *Timeseries* **Description**: Histograms may be used, for presentation purposes at the `Web Interface`, to display the activity levels through the requested period.
hasActivityLevelTimeseries	**Domain**: *ActivityLevelData* **Range**: *Timeseries* **Description**: Estimated physical activity level timeseries (categorical value).
hasActivitySensorStartEndTimes	**Domain:** *GlobalIndicator* **Range:** *Timeseries* **Description**: Global indicator for displaying the sensor start and end of usage times.
hasActivityTypeTimeseries	**Domain**: *ActivityTypeData* **Range**: *Timeseries* **Description**: Estimation of activity type. The timeseries describe the activity type over time or activity type/probability pairs.
hasAggregatedPhysicalActivity	**Domain**: *GlobalIndicator*

	Range: *Timeseries* **Description**: Global indicator for displaying aggregated physical activity graphs (e.g. aggregated per 5 minute spans).
hasAnswersForQuestionnaire	**Domain**: *QuestionnaireAnswers* **Range**: *Questionnaire* **Description**: The answers to the *Questionnaires* are stored separately. This relation indicates to which *Questionnaire* the answers for are.
hasAppointment	**Domain**: *PersonalisedGuidanceEvent*, *BehaviouralAssessmentEvent* **Range**: *Appointment* **Description**: Relation to indicate that *Appointment* may be assigned as a part of a `BAE` or a `PGE`.
hasAssistant	**Domain**: *PrimaryScreeningStage* **Range**: *Assistant* **Description**: Relation to indicate that one or more `Assistants` may be assigned to a `PSS`.
hasChewData	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Chew data timeseries that may be recorded as a part of a meal using the `Chewing Sensor`.
hasChewingSensorStartEndTimes	**Domain**: *GlobalIndicator* **Range**: *Timeseries* **Description**: Global indicator displaying when the `Subject` started and stopped using the chewing sensors for the requested time period.
hasCreatorProfessional	**Domain**: *Subject*, *BehaviouralAssessmentEvent*, *Appointment*, *Assistant*, *Permission*, *PersonalisedGuidanceEvent*, *PrimaryScreeningEvent* **Range**: *Professional* **Description**: Relation to indicate the creator (and, thus, the owner) of an `Event`, user, permission entry or appointment.
hasDay	**Domain**: *BehaviouralAssessmentEvent*, *PersonalisedGuidanceEvent*

	Range: *Day* **Description**: Relation to allow the description of the data and the calculated indicators in a daily basis.
hasEatingPauses	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Time series data containing the pauses during eating (e.g. the timestamp of the pause and its durations).
hasEnergyExpenditureTimeseries	**Domain**: *EnergyExpenditureData* **Range**: *Timeseries* **Description**: Time series data containing the estimated energy expenditure.
hasExerciseIntensity	**Domain**: *GlobalIndicator* **Range**: *Timeseries* **Description**: Global indicator displaying the intensity in each exercise session.
hasFoodAddition	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Time series data collected/calculated during the meal. This time series may indicate the weight of the food addition and the probability the food addition was correctly identified.
hasFoodStructure	**Domain**: *Meal* **Range**: *FoodStructure* **Description**: Relation to indicate the food structure identified for each meal.
hasPermissionFor	**Domain**: *Permission* **Range**: *Professional* **Description**: Relation to indicate that a `Professional` has specific permission level to a `PSS`, `PSE`, `BAE` or `PGE`.
hasGlobalIndicator	**Domain**: *Day* **Range**: *GlobalIndicator* **Description**: Relation to indicate the global indicators may be collected/calculated during a day of a `BAE` or a `PGE`.

hasGoal	**Domain**: *PersonalisedGuidanceEvent* **Range**: *Goal* **Description**: Relation to indicate the personalised behavioural goals prescribed by the Professional for a PGE.
hasLocalIndicator	**Domain**: *Day* **Range**: *LocalIndicator* **Description**: Relation to indicate for the local indicators (time series data) collected/extracted during a day of a BAE or a PGE.
hasMeal	**Domain**: *Day*, *PrimaryScreeningEvent*, *TrainingMeal* **Range**: *Meal* **Description**: Relation for the meal recorded as a part of PSE, BAE, PGE or as training meals during a PSS.
hasMealStartTimes	**Domain**: *GlobalIndicator* **Range**: *Timeseries* **Description**: Global indicator displaying time series data for the meal start times of the requested period.
hasPermission	**Domain**: *PrimaryScreeningStage*, *PersonalisedGuidanceEvent*, *BehaviouralAssessmentEvent*, *PrimaryScreeningEvent* **Range**: *Permission* **Description**: Relation to indicate that permissions to other Professional may be assigned for a PSS, PSE, BAE or PGE by the creator (owner) Professional.
hasPrimaryScreeningStage	**Domain**: *TrainingMeal*, *PrimaryScreeningEvent*. **Range**: *PrimaryScreeningStage*. **Description**: Indicates that the domain classes belong to a PSS. The main goal of a *PrimaryScreeningStage* is to group the PGEs of the same screening phase (e.g. same school and age). Additionally, the training meals, since they are anonymous, are related directly to a *PrimaryScreeningStage*.
hasProcessedMandometerData	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Time series data for the grams over time of the raw food intake curve after processing the signal

	recorded by the Mandometer© scale.
hasQuestion	**Domain**: *Questionnaire* **Range**: *Question* **Description**: Relations to indicate which questions belong to a questionnaire.
hasQuestionnaire	**Domain**: *PrimaryScreeningEvent, PersonalisedGuidanceEvent, Meal, BehaviouralAssessmentEvent* **Range**: *Questionnaire* **Description**: Relation to indicate that a specific questionnaire type is requested to be answered during a meal, a PSE, a BAE or a PGE.
hasQuestionnaireAnswers	**Domain**: *Meal, PersonalisedGuidanceEvent, BehaviouralAssessmentEvent, PrimaryScreeningEvent* **Range**: *QuestionnaireAnswers* **Description**: Relation to indicate that a questionnaire has been answered. The same questionnaire may be answered many times during an Event.
hasRawMandometerData	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Time series for the grams over time of the raw food intake curve as recorded by the Mandometer© scale.
hasRiskFinalisedBy	**Domain**: *PrimaryScreeningEvent, BehaviouralAssessmentEvent* **Range**: *Professional* **Description**: Relation to indicate the Professional that finalised the risk category of a PSE or a BAE. The owner Professional or other Professionals with appropriate permissions to the Event may finalise the risk category.
hasSatietyLevel	**Domain**: *Meal* **Range**: *Timeseries* **Description**: Time series containing data for the satiety level at different times during a meal.
hasSnackingTimeseries	**Domain**: *SnackingData*

	Range: *Timeseries* **Description**: Local indicator data for the detected snacking events and/or the probability they were correctly detected.
hasSubject	**Domain**: *Appointment*, *PersonalisedGuidanceEvent*, *PrimaryScreeningEvent*, *BehaviouralAssessmentEvent* **Range**: *Subject* **Description**: Relation to indicate that an appointment, a `PSE`, a `BAE` or a `PGE` refers to a specific `Subject`.
hasUsageActivitySensorTimeseries	**Domain**: *UsageActivitySensorData* **Range**: *Timeseries* **Description**: Local indicator containing time series data for the usage (or the probability of usage) of the `Activity Sensor` over time.
hasUsageChewingSensorTimeseries	**Domain**: *UsageChewingSensorData* **Range**: *Timeseries* **Description**: Local indicator containing time series data for the usage (or the probability of usage) of the `Chewing Sensor` over time.
hasWorkSchedule	**Domain**: *BehaviouralAssessmentEvent*, *PersonalisedGuidanceEvent* **Range**: *WorkSchedule* **Description**: Relation to indicate that the `Subject`'s working schedule has been described in the context of a `BAE` or a `PGE`.

2.3 Time series JSON format

Timeseries class was introduced is a generic wrapper class for time series data. Time series data are not directly used for goal definition or evaluation of the goal proximity, thus, the use of a generic class simplifies the ontology design without loss of descriptiveness. In this section, we propose a JSON format for data identified as time series.

First, the JSON format shall define a *preamble* part for the metadata of the collected time series. This could include: *signal type*, *sampling frequency*, *sensors* employed, *start timestamp* and *end timestamp*. The next part shall contain the actual time series: the timestamps and the signal or the extracted values of the indicators. Additionally, some local indicators may provide additional time series indicating the probability the values were calculated correctly. Thus, the *time_series* part of JSON shall contain, at least, the following values: *timestamps*, *values* and, *probabilities*.

An example of such JSON file follows:

```
{
    /* Comments on preamble part
    "type" is set among predefined types
    "sampling_rate" is given in seconds
    "samples" is the number of samples
    "sensors" is an array for the used sensors
    "start_timestamp" is the start time of the recording
    "end_timestamp" is the end time of the recording
    */
    "preamble":
    {
        "type":"ACTIVITY_TYPE",
        "sampling_rate":10,
        "samples":5
        "sensors":["ACTIVITY_SENSOR"],
        "start_timestamp":"2014-11-21 17:05:00",
        "end_timestamp":"2014-11-21 17:05:20"
    },
    /* Comments on timeseries part
    "values" is the array of the recorded/extracted values
    "probabilities" is the (optional) array with additional
     information about the quality of the samples
    */
    "time_series":
    {
        "values":[2, 2, 3, 3, 2],
        "probabilities":[0.85, 0.82, 0.41, 0.43, 0.8]
    }
}
```

In the example, the *preamble* part defines that the signal is for activity type, the `Activity Sensor` was used for the samples, and the sampling rate was at 10 samples/sec with a total of 5 samples is contained in the JSON. Finally, the start and end time stamps of the recorded time series are defined as well.

The *time_series* part first defines an array of timestamps for the recorded samples. Next, an array of time series values that describe the activity type are given. Finally, in this type of signal, an array with the probabilities indicating the confidence of the measured values is given.

2.4 Ontology for Communication Interface Design

The ontology can be used for design of the communication interface. Here, we consider the use of JSON files for the communication of the different system modules. Using JSON objects for data transfer has several advantages. It is has a clean syntax that can describe the data model of most problems efficiently. Since it is not a mark-up language (like XML), JSON provides high flexibility without the need to define new tags or attributes to represent the data. To this end, we propose JSON for storing and exchanging data and an example of using the ontology for defining the format of the JSON files is given.

The data properties and object properties as presented in the ontology can be employed for the definition of JSON objects for the communication of the systems (e.g. data transfer from the `Mobile Phone` to the server, data retrieval from the server, etc.).

As an example, we will demonstrate how the ontology can be used for the definition of a JSON file format for meals. First, the data properties of the *Meal* class are added as variables of the JSON object, e.g. average bite frequency, average chewing rate, etc. (Table 1).

Then, the object properties of *Meal* class (Table 4) that indicate the relations with the other classes can be used to complete the JSON object. JSON file formats are implicitly defined in the example and are named according to the object properties of the *Meal* class: *hasMandometerRawData*, *hasProcessedMandomterData*, *hasChewData*, *hasEatingPauses*, *hasFoodAddition*, *hasSatietyLevel*, *hasQuestionnaireAnswers*, and *hasFoodStructure*.

An example JSON file for a registered meal recorded using `Mandometer`© and `Chewing Sensor` follows:

```
{
    /*
    First, the data properties of the
    Meal class are directly described.
    */
    "averageBiteFrequency":0.5,
    "averageChewingRate":2.3,
    "averageFoodIntake":18.4,
    "biteSize":37.6,
    "chewingRateAcceleration":23.2,
    "estimateKcal":1640,
    "hasChewingSensor":true,
    "hasMandometerSensor"true,
    "initialBiteSize":24,
    "initialChewingRate":3,
    "initialFoodWeight":434,
    "isConfirmedSnack":false,
    "isRegisteredMeal":true,
    "mealCurveAlpha":0.4,
    "mealCurveBeta":1.2,
    "mealDetectionProbability":null,
    "meaDuration":344,
    "mealStartTime":"2015-02-24 13:05:23",
    "mealEndTime":"2015-02-24 13:10:57",
    "mealType":"LUNCH",
    "numberChewsBeforeSwallow":4.2,
    "numberOfFoodAdditions":0,
    "pauseLengthBeforeEating":10.3,
    "satietyFoodIntakeRatio":0.38,
    "totalFoodIntake":430,
    "varianceOfBiteSize":3.8,
    "weightOfLeftOvers":4,
    /*
    Then, the object properties related to Meal
    class are added and are described using
    Timeseries or other JSON file formats.
    */
    /*
    Mandometer raw data is a time series.
    The Timeseries example of the previous
    section is used.
    */
    "hasMandometerRawData":
    {
        "preamble":
        {
```

```
            "type":"MANDOMETER_RAW",
            "sampling_rate":4,
            "samples":86
            "sensors":["MANDOMETER"],
            "start_timestamp":"2015-02-24 13:05:23",
            "end_timestamp":"2015-02-24 13:10:57"
        },
        "timeseries":
        {
            "values":[434, 420, ...],
            /* instead of null could be undefined */
            "probabilities":null
        }
    },

    "hasProcessedMandomterData": { /* time series JSON object */ }
    "hasChewData": { /* time series JSON object*/ },
    "hasEatingPauses": { /* time series JSON object */ },
    "hasFoodAddition": { /* time series JSON object*/ },
    "hasSatietyLevel": { /* time series JSON object*/ },

    /* An array containing two questionnaire JSON objects*/
    "hasQuestionnaireAnswers":
    [
        {
            "questionnaire_id":11,
            "questionnaire_answers":["YES", "WATER", ...]
        },
        {
            "questionnaire_id":12,
            "questionnaire_answers":["NO", "2", ...]
        }
    ],

    /* An example JSON object for food structure data*/
    "hasFoodStructure":
    {
        "type":"SOLID",
        "isCrispy":true,
        "isChewy":false,
        "isWet":false
    }
}
```

In this example, first we used the data properties of the *Meal* class as direct properties of the JSON. Then, the object properties indicating the relations with other classes were employed and the relevant classes were added using other JSON objects.

To this end, we first used the time series format of Section 2.3 to add the time series data of the meal. Since multiple questionnaires can be requested for a meal (e.g. before meal, after meal), the *hasQuestionnaireAnswers* property consists of an array of JSON objects for each of the answered questionnaire. In the example, the format of the questionnaire JSON object contains its type (*questionnaire_id*), and therefore implicitly indicates the questions, and an array with the answers is stored in the *questionnaire_answers* property. This example JSON file format for storing questionnaire answers can be used at other parts of the system too, e.g. for storing questionnaire answers for a BAE or a PGE.

Finally, the *hasFoodStructure* property is defined a JSON object as well. Its properties could change depending on the type of the food (e.g. if the food structure is identified as solid the *isCrispy, isChewy* and *isWet* property shall be expected in the object structure whereas, if the food structure was liquid the *liquidType* property would be expected).

2.5 Ontology for Storage Model Design

The ontology describes the data model and can be used for the design of the system's storage. We suppose that a relational database will be used for the data storage. The following steps can be used for designing a relational database using the ontology:

1. Most of the ontology classes can be added directly as tables of the relational databases.
2. Classes that can be compactly described with JSON objects can be added directly as fields (columns) of the previous tables.
3. The classes' object properties describe the relations between them hence can be used to define the foreign keys between the database tables.

As an example, we demonstrate the design of a relational database to cover the storage needs of the *Primary Screening* phase according to Section 2.2.1 for the *Primary Screening* phase ontology classes and data properties, and Section 2.2.4 for the object properties.

First, we add the most important classes as tables: *PrimaryScreeningEvent*, *PrimaryScreeningStage*, *Meal*, *Professional*, *Assistant*, *Permission*, *Subject*, *TrainingMeal*, *QuestionnaireAnswers*, *Questionnaire*, and *Question*.

Then, we add the classes' data properties as table fields. The class *FoodStructure* which may be described by a JSON object and directly refers to a single meal was added as a field of *Meal* table.

Finally, the object properties are used to define the foreign keys of a relational database. For the object properties that imply many-to-many relation additional *cross-reference* tables were used: *PrimaryScreeningStage_Assistant*, *PrimaryScreeningStage_Permission*, *PrimaryScreeningStage_Permission*, *PrimaryScreeningEvent_QuestionnaireAnswers*, and *Meal_QuestionnaireAnswers*.

Figure 5 displays the resulting database schema. The data properties are the table fields marked with blue and the foreign keys are the fields marked with red.

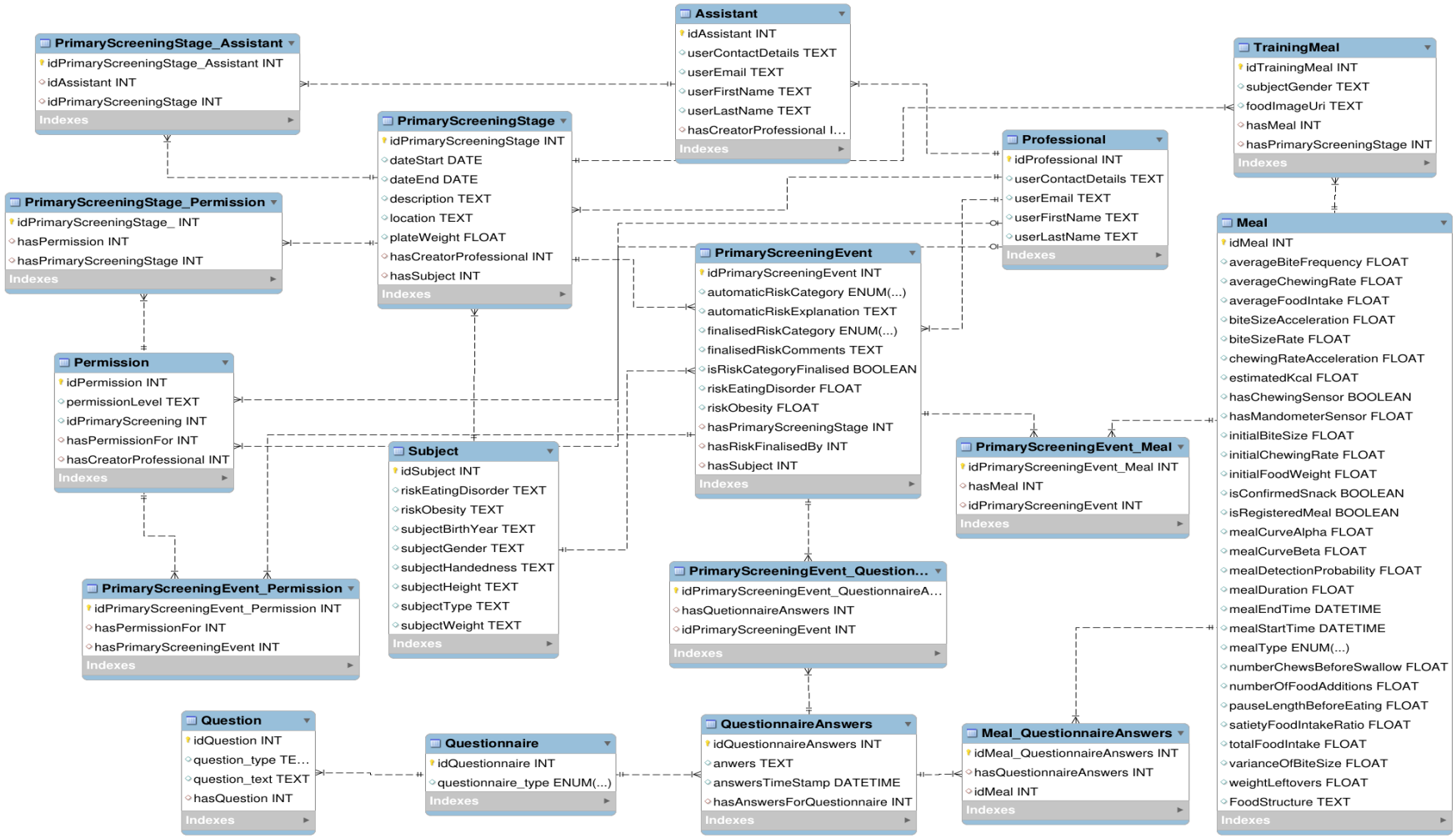

Figure 5 Example of database design using the ontology for the *Primary Screening* phase.

3 Goal Expression Language and Software Tool

During the *Personalised Guidance* phase, the `Professionals` must be able to prescribe personalised goals for the `Subjects`. This section aims to provide a formal language for the prescribed personalised goal is and a software tool for parsing such goals.

3.1 Goal Definition

The goals are defined in the following two forms:

- `SELECT_SQWRL::` *`select_sqwrl_statement`* `::END`
  `GOAL::` *`goal_statement`* `::END`

- `SELECT_SQL::` *`select_sql_statement`* `::END`
  `GOAL::` *`goal_statement`* `::END`

The goal consists of two parts: the select statement and the goal statement. There are two ways to define the select statement depending on the type of the query; either SQWRL or SQL which is denoted with `SELECT_SQWRL::` and `SELECT_SQL::` tags respectively. Then, the retrieved data are evaluated according to the goal statement (denoted with `GOAL::` tag). Section 3.1.1 describes the form of the select expressions and Section 3.1.2 presents the goal expressions.

3.1.1 Select statement

The *`select_sqwrl_statement`* (surrounded in `SELECT_SQWRL::` and `::END` tags) is an expression in SQWRL (Semantic Query-Enhanced Web Rule Language) [2]. SQWRL is a SWRL-based language for querying OWL ontologies. It provides SQL-like operations to retrieve knowledge from OWL. We chose SQWRL since it has a simple syntax and intuitive form that can be easily understandable. Furthermore, the SQWRL expressions can be easily transformed to other forms (e.g. to SQL expressions).

The returned result set of a SQWRL expression is a two dimensional table containing the results of the query (similar to the result sets returned from a relational database). For example, a *`select_sqwrl_statement`* to select the global indicators for a `Subject` with *subjectID* 45 for the working days can be:

```
PersonalisedGuidanceEvent(?pge) ^ hasSubject(?pge, ?subject) ^
hasDay(?pge, ?day) ^ hasGlobalIndicators(?day, ?glo_ind) ^
abox:hasValue(?subject, subjectID, 45) ^
abox:hasValue(?day, isWorkingDay, TRUE) -> sqwrl:select(?glo_ind)
```

The above expression can be directly used if the ontology is used as the system's knowledge base. However, even if the ontology is not used for data storage, SQWRL expressions remain an ideal way to express the data selection part of a goal in a formal way without concerning the underlying database implementation.

If a relational database is used for the data storage and the database schema is defined, then it may be easier to directly describe the selection statement of the goal using a SQL query. The goal selection allows the use of `SELECT_SQL::` and `::END`tags for this purpose. Similarly to the *`select_sqwrl_statement`*, the *`select_sql_statement`* returns a two dimensional table when the query is evaluated.

Let a relational database with schema containing the tables *PersonalisedGuidanceEvent* and *GlobalIndicators*. Then, the SQWRL expression of the previous example could be expressed in SQL as:

```
Select * FROM

(GlobalIndicators JOIN PersonalisedGuidanceEvent ON
GlobalIndicators.idPersonalisedGuidanceEvent =
PersonalisedGuidanceEvent.id)

WHERE PersonalisedGuidanceEvent.idSubject = 45 AND
GlobalIndicators.isWorkingDay = TRUE
```

3.1.2 Goal statement

The goal statement describes the goal based on the data columns returned from the select statement. Goal expressions can be formed using operators. Table 5 shows the relational and comparison operators. In addition, more complex goals can be defined by combining two or more expressions and obtain a single relational result using logical operators. Table 6 shows the available logical operators. The precedence of the expressions can be modified using parentheses.

Table 5 Relational and comparison operators available for Goal Expression

Operator	***Description***
==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

Table 6 Logical operators available for Goal Expression

Operator	***Description***
NOT	Boolean logical operation NOT. It has only one operand, to its right, and inverts it, producing `false` if its operand is `true` and `true` if its operand is `false`.

AND	Boolean logical operation AND. It has two operands and results to `true` if both its operands are `true` and `false` otherwise.
OR	Boolean logical operation AND. It has two operands and results to `true` if at least one operand is true and `false` otherwise.

The expressions can also contain *functions*. Most commonly, functions shall be used for manipulating the data retrieved from the goal's select statement, i.e. they receive their argument from the columns returned by the select statements. A function expression can replace the left-hand operand of a relation/comparison operator. For example, functions can operate on the values of time series data to calculate indicators which can be evaluated in the goal statements.

Although the exact list of the necessary functions was not known during the writing of this deliverable, the software tool is able to identify when an expression contains function calls and parse the function name and its arguments.

3.2 Example Goals and Software Tool Output

This section illustrates goal examples and the output of the UI tool that was designed for parsing the goal expressions. The UI tool parses the input goal expression, identifies the selection expression and the goal expression and constructs a parse tree for the goal expression. Annex A.1 provides the BNF definition and lexical analyser source code of the UI tool that was developed using Bison [3] as parser generator and Flex [4] as lexical analyser.

At this point it should be noticed that Bison gives provides us the capability to add code that evaluates the goal statements as the goal statement is parsed. In the final version of the goal assignment module the goal proximity of a `Subject` will be assigned by an updated version of the herein presented tool.

3.2.1 Goal Example 1

Description:

Request the lunch meals of `Subject` with identifier 45 to have food intake acceleration (mealCurveAlpha) 0.5 g/sec^2 and initial food intake rate (mealCurveBeta) -0.2 g/sec. The select statement is defined using SQWRL.

Goal expression:

```
SELECT_SQWRL::

PersonalisedGuidanceEvent(?pge) ^ hasSubject(?pge, ?subject) ^ hasDay(?pge, ?day) ^ hasMeal(?day, ?meal) ^ abox:hasValue(?subject, subjectID, 45) ^ abox:hasValue(?meal, mealType , "LUNCH") -> sqwrl:select(?meal)

::END

GOAL::

(mealCurveAlpha == 0.3) AND (mealCurveBeta == -0.2)

::END
```

UI tool output:

```
Select statement: PersonalisedGuidanceEvent(?pge) ^ hasSubject(?pge,
?subject) ^ hasDay(?pge, ?day) ^ hasMeal(?day, ?meal) ^
abox:hasValue(?subject, subjectID, 45) ^ abox:hasValue(?meal, mealType ,
"LUNCH") -> sqwrl:select(?meal)

GOAL 2

      Complex goal

      Left: GOAL 0

      Right: GOAL 1

      operator: AND

GOAL 0

      Left operand: mealCurveAlpha

      Right operand: 0.300000

      operator: ==

GOAL 1

      Left operand: mealCurveBeta

      Right operand: -0.200000

      operator: ==
```

The parser identifies the top goal (`GOAL 2`) of the parse tree as a complex goal consisting of two sub-goals (`GOAL 0` and `GOAL 1`) combined with the logical operator `AND`. The parsing continues and the sub-goals are parsed as final leafs of the parse tree.

3.2.2 Goal Example 2

Description:

Request the aggregated activity level to be over 3 on the days-off for time between 17.00 and 19.00. Now the select statement is defined using SQL and we assume that there is a table *ActivityLevelData* in the relational database containing the local indicators for the activity level.

Goal expression:

```
SELECT_SQL::

SELECT activity_level, day FROM ActivityLevelData WHERE
ActivityLevelData.isWorkingDay == FALSE AND ActivityLevelData.idSubject ==
45 AND ActivityLevelData.startTime > "17:00:00" AND
ActivityLevelData.endTime < "19:00:00" GROUP BY day

::END

GOAL::

aggregatedActivityLevel(activity_level) > 3

::END
```

Parser Output:

```
Select statement: SELECT activity_level, day FROM ActivityLevelData WHERE
ActivityLevelData.isWorkingDay == FALSE AND ActivityLevelData.idSubject ==
45 AND ActivityLevelData.startTime > "17:00:00" AND
ActivityLevelData.endTime < "19:00:00" GROUP BY day
```

```
GOAL 0
      Left operand: function "aggregatedActivityLevel" with argument:
      activity_level
      Right operand: 3
      operator: >
```

The tool identifies a single goal with left operand to be an expression containing a function. More specifically, the function is named `aggregatedActivityLevel` and operates on the retrieved data of the column `activity_level`. The results of the function call shall be calculated before the `GOAL 0` is evaluated.

3.2.3 Goal Example 3

Description: An illegal expression is given to the parser.

Goal expression:

```
SELECT_SQWRL::
PersonalisedGuidanceEvent(?pge) ^ hasSubject(?pge, ?subject) ^ hasDay(?pge,
?day) ^ hasMeal(?day, ?meal) ^ abox:hasValue(?subject, subjectID, 45) ^
abox:hasValue(?meal, mealType , "LUNCH") sqwrl:select(?meal)
::END
GOAL::
(mealCurveAlpha == 0.3) AND (mealCurveBeta == -0.2
::END
```

Parser Output:

```
syntax error
```

The goal expression is not complete, since there is a missing left parenthesis at the second sub-goal, and the parser yields an error message.

4 Conclusions

The present report, entitled "Formal ontology for the eating and activity behaviour domain", has documented the SPLENDID ontology for the eating and activity behaviour domain, has provided the definition of a formal language for the prescription of personalised goals, and has documented the usage of a UI tool for validating goal expressions.

First, the ontology was designed based on the UCs of D1.2 taking into consideration that it should accurately describe the data model of SPLENDID system and allow accurate description of the personalised behavioural goals. Furthermore, examples for using the ontology as basis for system's data model and communication interface design have been presented.

Next, the formal language for goal expression was defined. The goal language allows the definition of complex goals that consist of statements for retrieving the data and evaluating the goal. The flexibility of the goal evaluation is achieved using various logical and comparison operators along with the capability of using functions for data manipulation. A UI tool for parsing behavioural goals is presented that allows syntactic evaluation and parsing of the goals and examples are provided to demonstrate its functionality.

Concluding, we argue that the ontology and UI tool provide the groundwork for the final modules that will be included in D3.3 and D3.4. Moreover, the present report can be of use by the development teams of SPLENDID for the data model and communication interface design.

References

[1] Crockford, Douglas. "The application/json media type for javascript object notation (JSON)." (2006).

[2] O'Connor, Martin J., and Amar K. Das. "SQWRL: A Query Language for OWL." *OWLED*. Vol. 529. 2009.

[3] Donnelly, Charles, and Richard Stallman. "Bison. The YACC-compatible Parser Generator." (2004).

[4] Paxson, Vern. "Flex–Fast lexical analyzer generator." *Lawrence Berkeley Laboratory* (1995).

A. Annex A – Software tool for goal expression

A.1 Source code

A.1.1 BNF used with Bison

```
%%
goal_expression: select_section goal_section
{ $$ = $2; }
;
select_section: SELECTSQL {
    select_statement = $1;
    $$ = $1;
}
| SELECTSQWRL {
    select_statement = $1;
    $$ = $1;
}
;
goal_section: GOAL goal END { root = $2; $$ = $2; }
function_call: IDENTIFIER LPAREN fn_arguments RPAREN
{
    current_goal = init_goal();
    current_goal->lval.function = $1;
    current_goal->lval.field = $3;
    $$ = $1;
}
| IDENTIFIER LPAREN RPAREN
{
    current_goal = init_goal();
    current_goal->lval.function = $1;
    current_goal->lval.field = NULL;
    $$ = $1;
};
fn_arguments: IDENTIFIER { $$ = $1; }
| STRING { $$ = $1; }
;
gen_field: IDENTIFIER {
    current_goal = init_goal();
    current_goal->lval.field = $1;
}
| function_call { $$ = $1; }
;
goal: LPAREN goal RPAREN { $$ = $2; }
| NOT goal {
    current_goal = init_goal();
    current_goal->goalop = not;
    current_goal->left = NULL;
    current_goal->right = $2;
    $$ = current_goal;
}
| goal AND goal {
    current_goal = init_goal();
    current_goal->goalop = and;
    current_goal->left = $1;
    ($1)->parent = current_goal;
    current_goal->right = $3;
    ($3)->parent = current_goal;
```

```
    $$ = current_goal;
}
| goal OR goal {
    current_goal = init_goal();
    current_goal->goalop = or;
    current_goal->left = $1;
    ($1)->parent = current_goal;
    current_goal->right = $3;
    ($3)->parent = current_goal;
    $$ = current_goal;
}
| gen_field OPERATOR FLOAT {
    current_goal->rval.rval_double = $3;
    current_goal->valop = value_operator($2);
    $$ = current_goal;
}
| gen_field OPERATOR INTEGER  {
    current_goal->rval.rval_int = $3;
    current_goal->valop = value_operator($2);
    $$ = current_goal;
}
| gen_field OPERATOR STRING {
    current_goal->rval.rval = $3;
    current_goal->valop = value_operator($2);
    $$ = current_goal;
}
;
%%
```

A.1.2 Lexical analyser written in Flex

```
%{
#include "goal.h"
#include "goal_expression.tab.h" /* The tokens */

#define MAX_STR_LENGTH 32768
static char strbuf[MAX_STR_LENGTH];

static void append(char *dest, const char *source)
{
    if (strlen(dest) + strlen(source) > MAX_STR_LENGTH - 1) {
        return;
    }
    strcat(dest, source);
}
%}

%x string

DIGIT [0-9]
ID [a-zA-Z0-9_][a-zA-Z0-9_\-]*
INTEGER {DIGIT}+
FLOAT [\-]{DIGIT}+"."{DIGIT}*|{DIGIT}+"."{DIGIT}*
NUMBER {INTEGER}|{FLOAT}
SELECTSQL "SELECT_SQL::".+"::END"
SELECTSQWRL "SELECT_SQWRL::".+"::END"
OPERATOR "=="|"!="|">"|"<"|">="|"<="

%%
```

```
{SELECTSQL} { yylval.strval = strndup(&(yytext[13]), MAX_STR_LENGTH);
yylval.strval[strlen(&(yytext[13])) - 5] = '\0'; return(SELECTSQL); }
{SELECTSQWRL} { yylval.strval = strndup(&(yytext[15]), MAX_STR_LENGTH);
yylval.strval[strlen(&(yytext[15])) - 5] = '\0'; return(SELECTSQWRL); }
"GOAL::" { return(GOAL); }
"::END" { return(END); }
"AND" { return(AND); }
"OR" { return(OR); }
"NOT" { return(NOT); }
{OPERATOR} { yylval.strval = strndup(yytext, MAX_STR_LENGTH);
return(OPERATOR); }
{INTEGER} { yylval.intval = atoi(yytext); return(INTEGER); }
{FLOAT} { yylval.doubleval = atof(yytext); return(FLOAT); }
{ID} { yylval.strval = strndup(yytext, MAX_STR_LENGTH); return(IDENTIFIER);
}
"(" { return(LPAREN); }
")" { return(RPAREN); }
\"                  { BEGIN string; strbuf[0] = '\0'; }
<string>[^\\"\n]*   { append(strbuf, yytext); }
<string>\\n         { append(strbuf, "\n"); }
<string>\\t         { append(strbuf, "\t"); }
<string>\"          { yylval.strval = strdup(strbuf); BEGIN 0; return
STRING; }
<string>\\.         { fprintf(stderr, "bogus escape '%s' in string\n",
yytext); }
<string>\n          { fprintf(stderr, "newline in string\n"); }
[ \t\n\r]+  /* eat up whitespace */
. { return(yytext[0]); }
%%

/* Additional C code */
int yywrap(void) {}
```