POWER2DM

“Predictive model-based decision support for diabetes patient empowerment”

Research and Innovation Project
PHC 28 – 2015: Self-management of health and disease and decision support systems based on predictive computer modelling used by the patient him or herself

POWER2DM D3.6 (or D3.3.1b)

Action Plan Engine II

Due Date:	31th October 2018 (M33)
Actual Submission Date:	31th October 2018
Project Dates:	Project Start Date: February 01, 2016
	Project End Date: July 31, 2019
	Project Duration: 54 months
Deliverable Leader:	SRFG

Project co-funded by the European Commission within H2020 Programme (20015-2016)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Document History:

Version	Date	Changes	From	Review
V0.1	2017-11-13	Implementation report template and content structure based on D3.5	SRFG	
V0.2	2017-12-11	Distribution of Work	SRFG	
V0.3	2018-08-02	Identify missing pieces	SRFG	
v0.4	2018-09-28	Adding Action Plan Deployment	SRFG	
v0.5	2018-10-20	Migrate Security + Action Plan Services to v2	SRFG	
v0.6	2018-10-29	Updates in the UI section	SRFG	
v1.0	2018-10-31	Final integrations, corrections and review.	SRFG	

Contributors (Benef.) Felix Strohmeier (SRFG)
Dietmar Glachs (SRFG)
Oliver Jung (SRFG)
Robert Mulrenin (SRFG)

Responsible Author Felix Strohmeier (SRFG) **Email** felix.strohmeier@salzburgresearch.at

POWER2DM Consortium Partners

Abbv	Participant Organization Name	Country
TNO	Nederlandse Organisatie voor Toegepast Natuurwetenschappelijk Onderzoek	Netherlands
IDK	Institute of Diabetes “Gerhardt Katsch” Karlsburg	Germany
SRDC	SRDC Yazilim Arastirma ve Gelistirme ve Danismanlik Ticaret Limited Sirketi	Turkey
LUMC	Leiden University Medical Center	Netherlands
SAS	SAS Servicio Andaluz de Salud	Spain
SRFG	Salzburg Research Forschungs Gesellschaft	Austria
PD	PrimeData	Netherlands
iHealth	iHealth EU	France

TABLE OF CONTENTS

EXECUTIVE SUMMARY

In the scope of Task 3.3 “Action Plan Engine” the Frontend and Backend modules of the Web-based module in POWER2DM were designed and implemented. This document describes the general architecture, major components and the technological pillars used for the implementation of the Web-Application of the self-management support system for the patients. The document comprises the following sections:

Chapter 2 provides an overall architecture of the POWER2DM architecture, the internal design of the Action Plan Engine and the technology foundation used for this part of the project. It also shows how the Action Plan Engine is embedded in the overall project context.

Chapter 3 contains different aspects in the implementation:

- the security integration using a single-sign-on service for all components provided by POWER2DM,
- the integration with the storage solution used in POWER2DM (Personal Data Store - “PDS”),
- the REST-API that can are used by the GUI components and other front-end service, such as the mobile app,
- a workflow tool provided as authoring tool for process models,
- and the integration of the knowledge models developed in task T3.1.

Finally, Chapter 4 describes the project setup and deployment procedure used for delivering the current prototype releases to the cloud environment for the pilots managed by PrimeData and hosted by TNO.

1 Introduction

1.1 Purpose and Scope

In the scope of Task 3.3 the Action Plan Engine has been developed. The purpose of deliverable D3.6 is to provide the software implementation and demonstrator for POWER2DM Action Plan Engine. This document provides an implementation report illustrating its architecture, technological background and gives an overview of the provided functionality for the prototypes deployed in the Pilot Studies in Spain and the Netherlands.

1.2 References

- D1.2 Requirement Analysis of POWER2DM
- D1.3 Conceptual Design of POWER2DM
- D3.2 Dynamic Health Behaviour Change Intervention Models for Self-management II
- D3.4 Communication Engine II
- D3.7 Mock-ups for Web and Mobile User Interfaces for SMSS Interventions
- D3.8 Web-Based GUI Components for DSS
- D3.9 Mobile GUI Components for DSS
- D4.1 Personal Data Model and Service API
- D4.2 Personal Data Store Service Implementation
- D4.9 Privacy/Security Enablers for POWER2DM Services - I

2 Approach, Architecture and Data Models

2.1 Overview

The Action Plan Engine is designed as a Self-Management-Support-System (SMSS) fostering behavior changes for diabetic patients. As shown in Figure 1, the Action Plan Engine is developed as part of the POWER2DM SMSS.

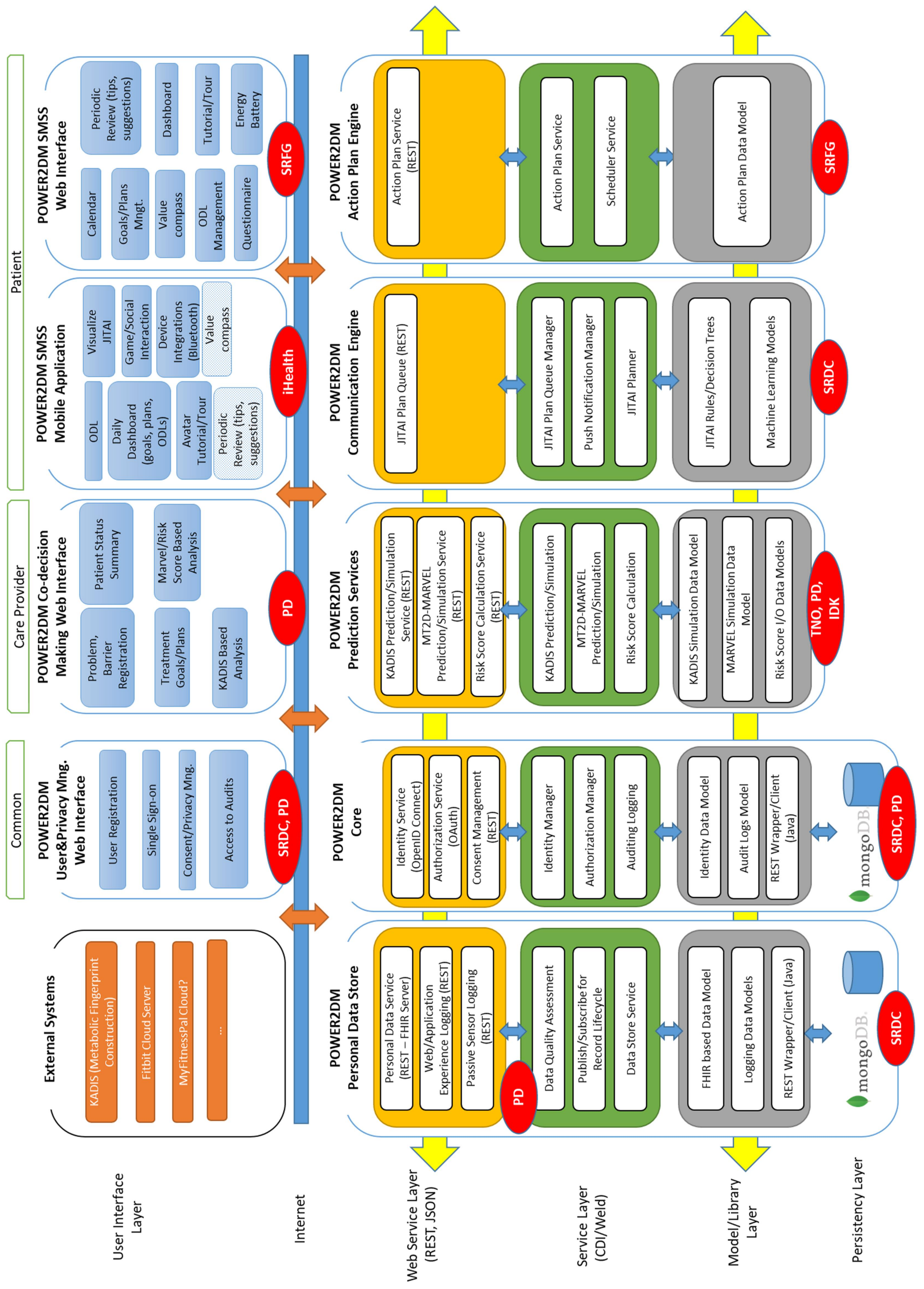

Patient
Care Provider
Common
POWER2DM SMSS Web Interface
Calendar
Goals/Plans Mngt.
Value compass
ODL Management
Questionnaire
Periodic Review (tips, suggestions)
Dashboard
Tutorial/Tour
Energy Battery
SRFG
POWER2DM SMSS Mobile Application
ODL
Daily Dashboard (goals, plans, ODLs)
Avatar Tutorial/Tour
Periodic Review (tips, suggestions)
Visualize JITAI
Game/Social Interaction
Device Integrations (Bluetooth)
Value compass
iHealth
POWER2DM Co-decision Making Web Interface
Problem, Barrier Registration
Treatment Goals/Plans
KADIS Based Analysis
Patient Status Summary
Marvel/Risk Score Based Analysis
PD
POWER2DM User&Privacy Mng. Web Interface
User Registration
Single Sign-on
Consent/Privacy Mng.
Access to Audits
SRDC, PD
External Systems
KADIS (Metabolic Fingerprint Construction)
Fitbit Cloud Server
MyFitnessPal Cloud?
...
POWER2DM Action Plan Engine
Action Plan Service (REST)
Action Plan Service
Scheduler Service
Action Plan Data Model
SRFG
POWER2DM Communication Engine
JITAI Plan Queue (REST)
JITAI Plan Queue Manager
Push Notification Manager
JITAI Planner
JITAI Rules/Decision Trees
Machine Learning Models
SRDC
POWER2DM Prediction Services
KADIS Prediction/Simulation Service (REST)
MT2D-MARVEL Prediction/Simulation Service (REST)
Risk Score Calculation Service (REST)
KADIS Prediction/Simulation
MT2D-MARVEL Prediction/Simulation
Risk Score Calculation
KADIS Simulation Data Model
MARVEL Simulation Data Model
Risk Score I/O Data Models
TNO, PD, IDK
POWER2DM Core
Identity Service (OpenID Connect)
Authorization Service (OAuth)
Consent Management (REST)
Identity Manager
Authorization Manager
Auditing Logging
Identity Data Model
Audit Logs Model
REST Wrapper/Client (Java)
mongoDB
SRDC, PD
POWER2DM Personal Data Store
Personal Data Service (REST – FHIR Server)
Web/Application Experience Logging (REST)
Passive Sensor Logging (REST)
PD
Data Quality Assessment
Publish/Subscribe for Record Lifecycle
Data Store Service
FHIR based Data Model
Logging Data Models
REST Wrapper/Client (Java)
mongoDB
SRDC
User Interface Layer
Internet
Web Service Layer (REST, JSON)
Service Layer (CDI/Weld)
Model/Library Layer
Persistency Layer

Figure 1: POWER2DM Conceptual Design

The Action Plan Engine represents a single pillar in the POWER2DM Conceptual Design and adds the Action Plan Engine GUI components on the user interface layer to the entire POWER2DM SMSS. The Action Plan Engine GUI provides all components POWER2DM SMSS Web Interface for Patient Self-Management, including Calendar, Goals and Plans Management, Value Compass, etc.

The POWER2DM user interface uses different web services, while the Action Plan Service represents the dedicated service endpoint for the Action Plan Engine GUI. Figure 1 outlines the distinct modules of the POWER2DM SMSS. Furthermore it also outlines the use of Internet technologies for service communication (REST/JSON) and the use of Dependency Injection for service discovery (CDI/Weld in the case of the Action Plan Engine). These technologies as well as the internal structure of the Action Plan Engine are explained in detail in the following sections.

2.2 Technology Foundation

External software dependencies have been carefully selected to bring in existing functionality with as little effort as possible, while still avoiding lock-in effects. Therefore, software products released under permissive open source licenses have been preferred over commercial third party software. In this section we describe the major libraries and frameworks used for the implementation of the Action Plan Engine and User Interfaces.

The Action Plan Engine is developed using Java Enterprise (JEE[1]) technologies. It requires a Servlet Container such as Apache Tomcat[2] or Jetty[3] as runtime environment. For the dependency injection the reference implementation CDI/Weld[4] is used. For the build and dependency management, Maven[5] as the de-facto standard is used to configure the distinct sub-components to be compiled into the final Action Plan Engine distribution.

The REST/JSON based web services for the user interface (UI) are implemented using the Jersey RESTful Web Services framework. A security layer based on Apache Shiro[6] is used to integrate with external authentication services such as OAuth to ensure that only authenticated and authorized users can access private resources. A detailed example for secure access to the PDS Resources is provided in Section 3.3.

The GUI is implemented by means of a HTML5/CSS3/JS single-page web application focusing on a responsive, user-centered design. Native HTML5 components, such as web storage, are used for higher level functionality. The following open-source JS libraries are included in order to assure proper cross-browser support of complex functionality:

- Bootstrap: Responsive design framework (MIT license)
  http://getbootstrap.com/
- Fullcalendar: Calendar plugin (MIT license)
  https://fullcalendar.io/
- i18next: Internationalization (MIT license)
  http://i18next.com/
- jqPlot: Plotting and charting (MIT license)

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html

[2] http://tomat.apache.org

[3] https://www.eclipse.org/jetty/

[4] http://weld.cdi-spec.org/

[5] http://maven.apache.org/

[6] https://shiro.apache.org/

http://www.jqplot.com/

- jQuery: Multi-feature framework (MIT license)
  https://jquery.com/
- jsSHA: Client-side hashing (BSD license)
  https://caligatio.github.io/jsSHA/
- Marked: Markdown syntax support (MIT license)
  http://www.javascriptoo.com/marked
- momentJS: Date and time validation and manipulation (MIT license)
  https://momentjs.com/

2.3 Internal Structure and Data Model

The Action Plan Engine adheres to principles and technologies as shown above. It provides the backend intelligence service for POWER2DM SMSS Mobile and Web applications that helps patient manage his/her self-management goals and action plans. It contributes the Action Plan Data Model to the Model/Library Layer of the POWER2DM SMSS. It implements requested functionality to provide interventions based on the collected ODLs e.g. feedback on planned activities, motivational messages and tips for fostering or improving self-management activities. Finally, it exposes its functionality via REST Web Services to the GUI Layer of POWER2DM SMSS providing JSON encoded data to be used either in the Action Plan Engine GUI or in the Mobile Application.

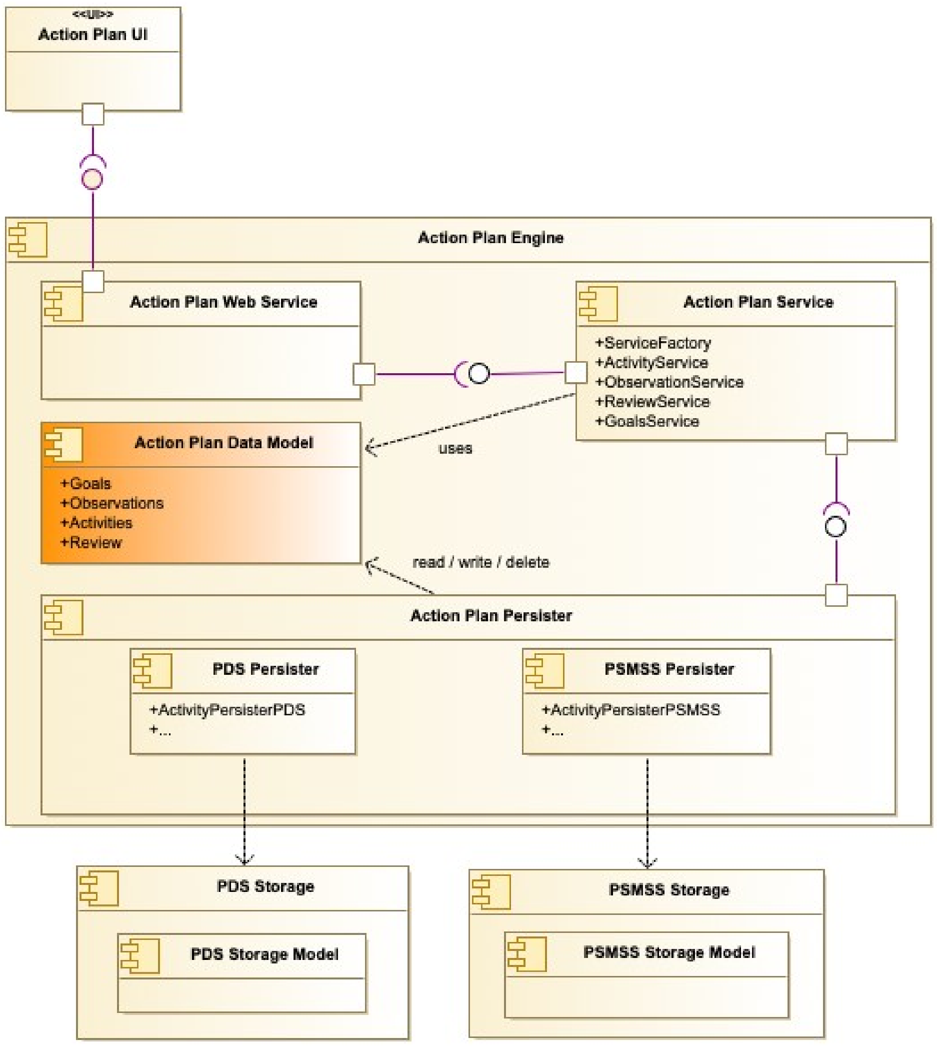


Figure 2: Action Plan Engine Internal Structure

A more detailed component overview of the Action Plan Engine is given in Figure 2. As shown, the Action Plan GUI communicates with its dedicated backend component. This backend component implements the required functionality to answer all requests from the GUI.

In addition, the Action Plan Engine defines its own internal data model for applying the algorithms for matching activities to observations, describing schedules, calculating calendar events and providing reviews for self-assessment by the patient. The entire model for the Action Plan Engine comprises the main types as outlined in Figure 3.

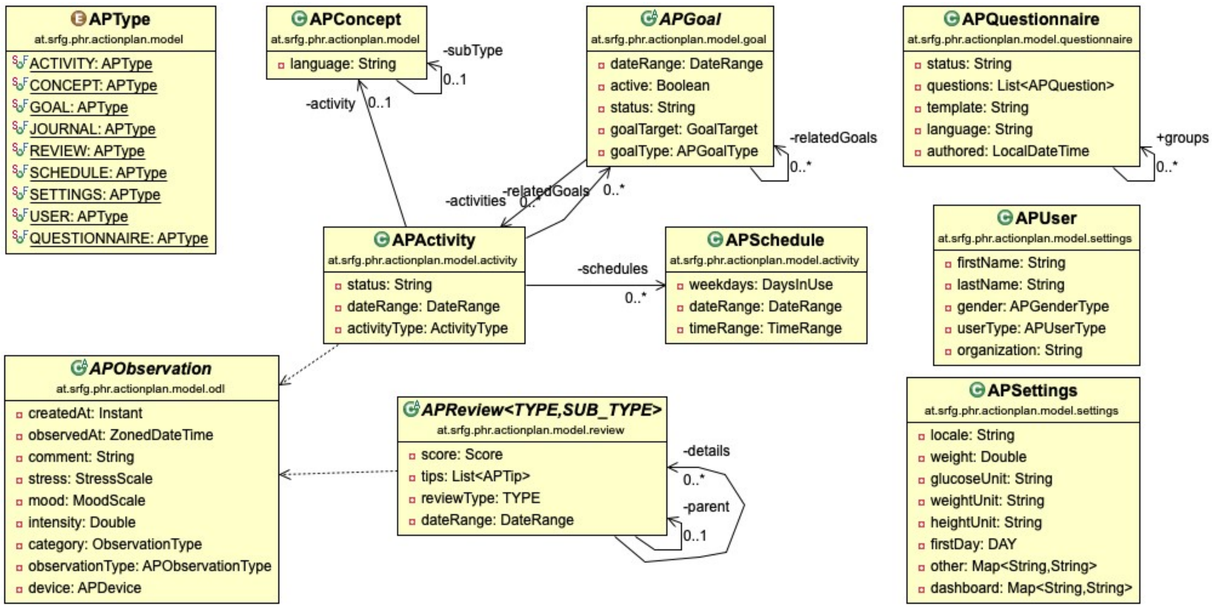


Figure 3: Action Plan Data Model

The main aspects of the Action Plan Engine Data Model covers:

- Goals, Treatment Goals and Planned Activities,
- Observations,
- Review,
- Energy Battery, and
- Questionnaires.

Each of those concepts is described in more detail in the following subsections.

2.3.1 Goals, Treatment Goals and Activities

This section covers the treatment goals issued by the care providers to the patient and the self-management adoption of the treatment goals by the patient. The patient may additionally express his/her own goals and create an action plan in order to achieve the expressed goals. The related data model objects are outlined in Figure 4.

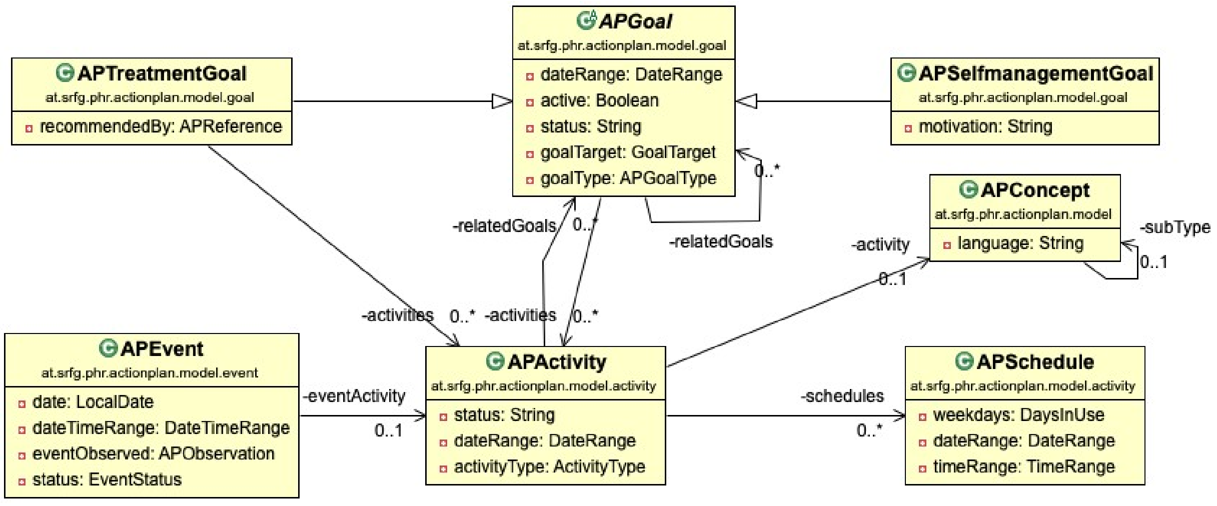


Figure 4: Action Plan Engine Data Model - Goals, Activities

Each activity such as “Measure Blood glucose” is accompanied with a set of schedules to determine the exact date & time when the patient is supposed to execute the activity. The resulting combination of the activity (planned action) and the schedule (when to perform) is expressed as events finally shown to the patient as calendar. An event is always linked to an activity which in turn may be assigned to multiple goals.

2.3.2 Observations

The action plan outlines the planned activities and the scheduled events. Whenever a patient effectively performs a planned activity, this is recorded as an observation. To reflect the distinct observation types – doing exercises, measuring weight, glucose or pressure, or simply observing sleep, stress – the distinct sub-classes shown in Figure 5 of the (abstract) observation are in use.

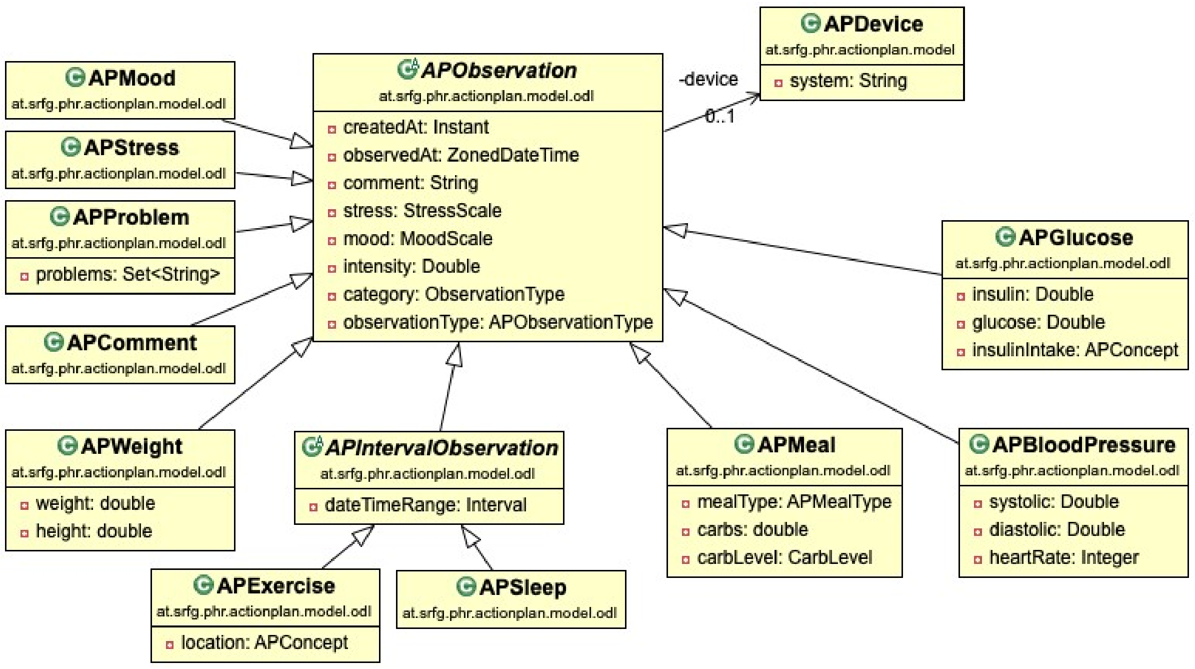

APObservation
at.srfg.phr.actionplan.model.odl
createdAt: Instant
observedAt: ZonedDateTime
comment: String
stress: StressScale
mood: MoodScale
intensity: Double
category: ObservationType
observationType: APObservationType
-device
0..1
APDevice
at.srfg.phr.actionplan.model
system: String
APMood
at.srfg.phr.actionplan.model.odl
APStress
at.srfg.phr.actionplan.model.odl
APProblem
at.srfg.phr.actionplan.model.odl
problems: Set<String>
APComment
at.srfg.phr.actionplan.model.odl
APGlucose
at.srfg.phr.actionplan.model.odl
insulin: Double
glucose: Double
insulinIntake: APConcept
APWeight
at.srfg.phr.actionplan.model.odl
weight: double
height: double
APIntervalObservation
at.srfg.phr.actionplan.model.odl
dateTimeRange: Interval
APMeal
at.srfg.phr.actionplan.model.odl
mealType: APMealType
carbs: double
carbLevel: CarbLevel
APBloodPressure
at.srfg.phr.actionplan.model.odl
systolic: Double
diastolic: Double
heartRate: Integer
APExercise
at.srfg.phr.actionplan.model.odl
location: APConcept
APSleep
at.srfg.phr.actionplan.model.odl

Figure 5: Action Plan Engine Data Model - Observations

This approach allows a tailored data exchange with the GUI, only the required data elements are exposed to the GUI developers or other service users.

2.3.3 Review

The patient is informed about his/her adherence to the planned activities as well as the achievement of the expressed goals with the review. This section provides an overview (including interventions or tips) whether the goals have been reached or should be adopted for the next period of time. The review is computed at an overall basis, e.g. all observations are matched against the action plan to compute the `Score` object outlining the number of planned and completed events. More details on the concept of review scores and corresponding interventions are described in Section 3.8.

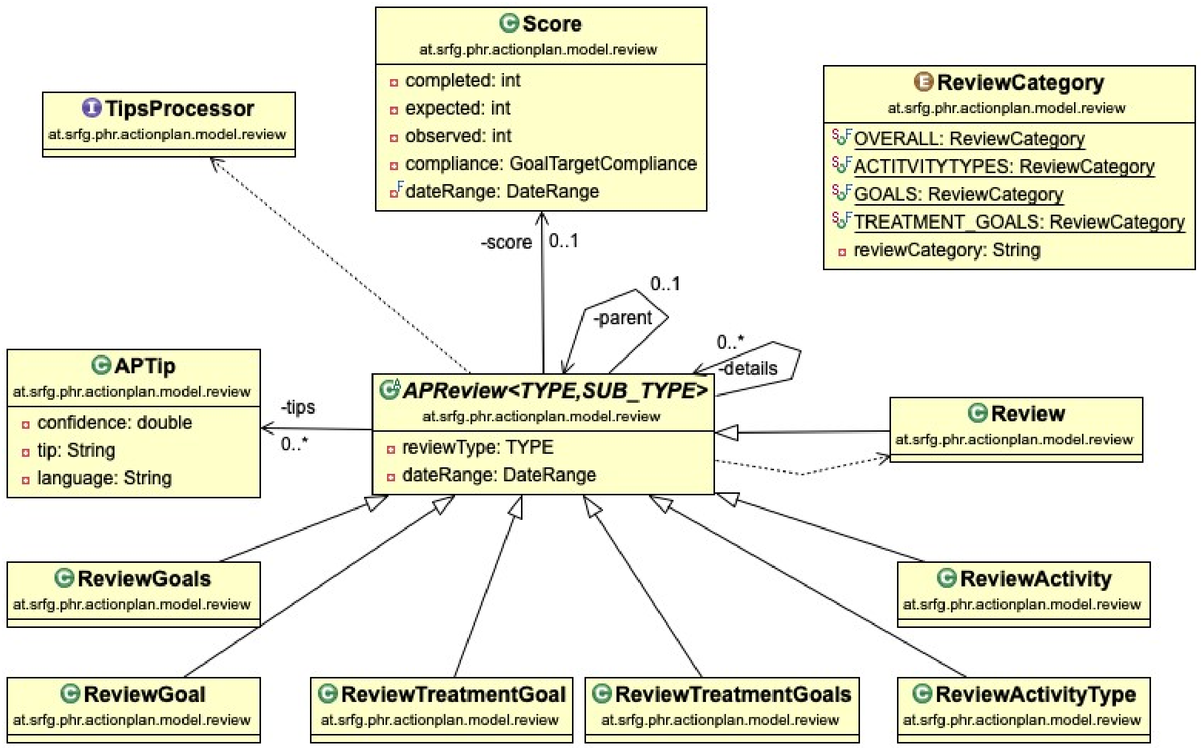

Score
at.srfg.phr.actionplan.model.review
completed: int
expected: int
observed: int
compliance: GoalTargetCompliance
dateRange: DateRange
TipsProcessor
at.srfg.phr.actionplan.model.review
ReviewCategory
at.srfg.phr.actionplan.model.review
OVERALL: ReviewCategory
ACTITVITYTYPES: ReviewCategory
GOALS: ReviewCategory
TREATMENT_GOALS: ReviewCategory
reviewCategory: String
-score
0..1
0..1
-parent
0..*
-details
APTip
at.srfg.phr.actionplan.model.review
confidence: double
tip: String
language: String
-tips
0..*
APReview<TYPE,SUB_TYPE>
at.srfg.phr.actionplan.model.review
reviewType: TYPE
dateRange: DateRange
Review
at.srfg.phr.actionplan.model.review
ReviewGoals
at.srfg.phr.actionplan.model.review
ReviewActivity
at.srfg.phr.actionplan.model.review
ReviewGoal
at.srfg.phr.actionplan.model.review
ReviewTreatmentGoal
at.srfg.phr.actionplan.model.review
ReviewTreatmentGoals
at.srfg.phr.actionplan.model.review
ReviewActivityType
at.srfg.phr.actionplan.model.review

Figure 6: Action Plan Engine Data Model - Review

The action plan might additionally link to self-management and treatment goals. These relationships are covered in the review as details or sub-type where each sub-type only captures relevant observations, thus computes its own `Score`. Figure 6 shows the data model objects related to the review.

2.3.4 Energy Battery

The energy battery is a tool allowing the patient to identify tasks and activities which are well appreciated – thus have a positive effect to the patient's feelings and behavior. Consequently, the battery also helps to identify tasks and activities which cause low mood, much stress or sleeping problems. The data model required for the energy battery is shown in Figure 7.

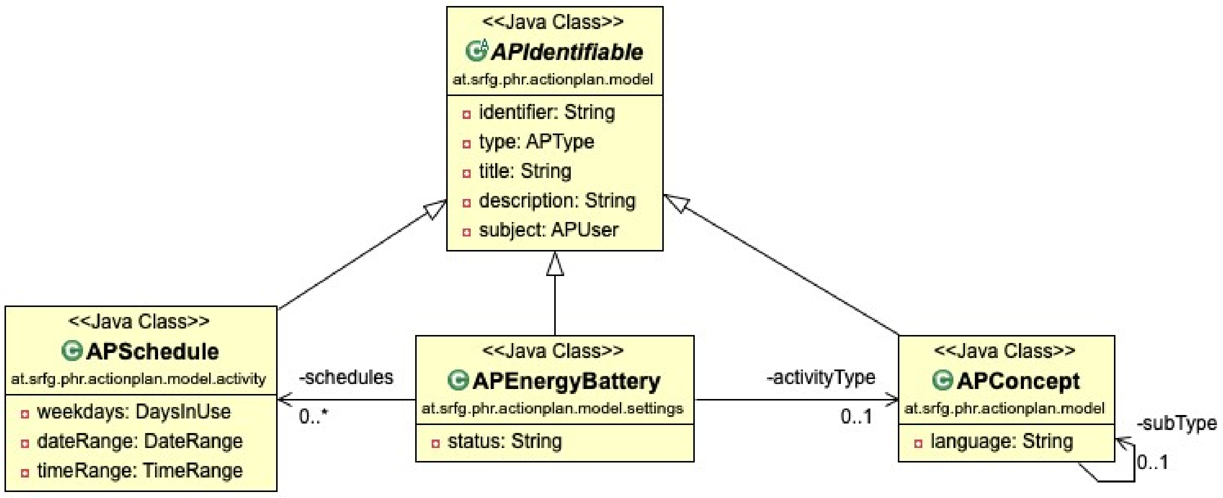


Figure 7: Action Plan Engine Data Model - Energy Battery

This allows the patient to classify his/her tasks whether they are active or passive relaxation or whether they are (exhaustive) Must Do actions.

2.3.5 Questionnaire

The patient is requested to fill out questionnaires after given period of times. A questionnaire may contain several questions where each question provides the possible answer options including the selected option as shown in Figure 8.

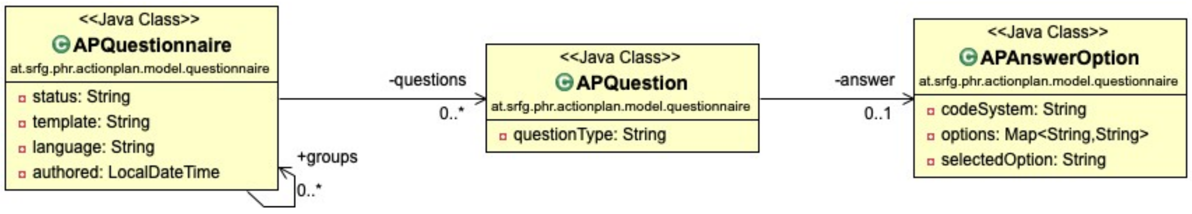


Figure 8: Action Plan Engine Data Model – Questionnaires

In case the patient answers a questionnaire the first time, the selected answer is empty. However, whenever the patient wants to amend his questionnaire result, the questionnaire is provided to the user with his/her selected answers.

3 Action Plan Implementation

3.1 Project Structure Overview

The Action Plan Engine uses Maven as its build management. The Project Object Model (POM) in the project's root folder is used when building the distribution and defines the required dependencies and sub modules to be included for each build. Maven supports a hierarchical structure of the modules and also allows the integration of pre-existing dependencies and libraries.

The current structure of the Action Plan Engine distribution is given in Figure 9.

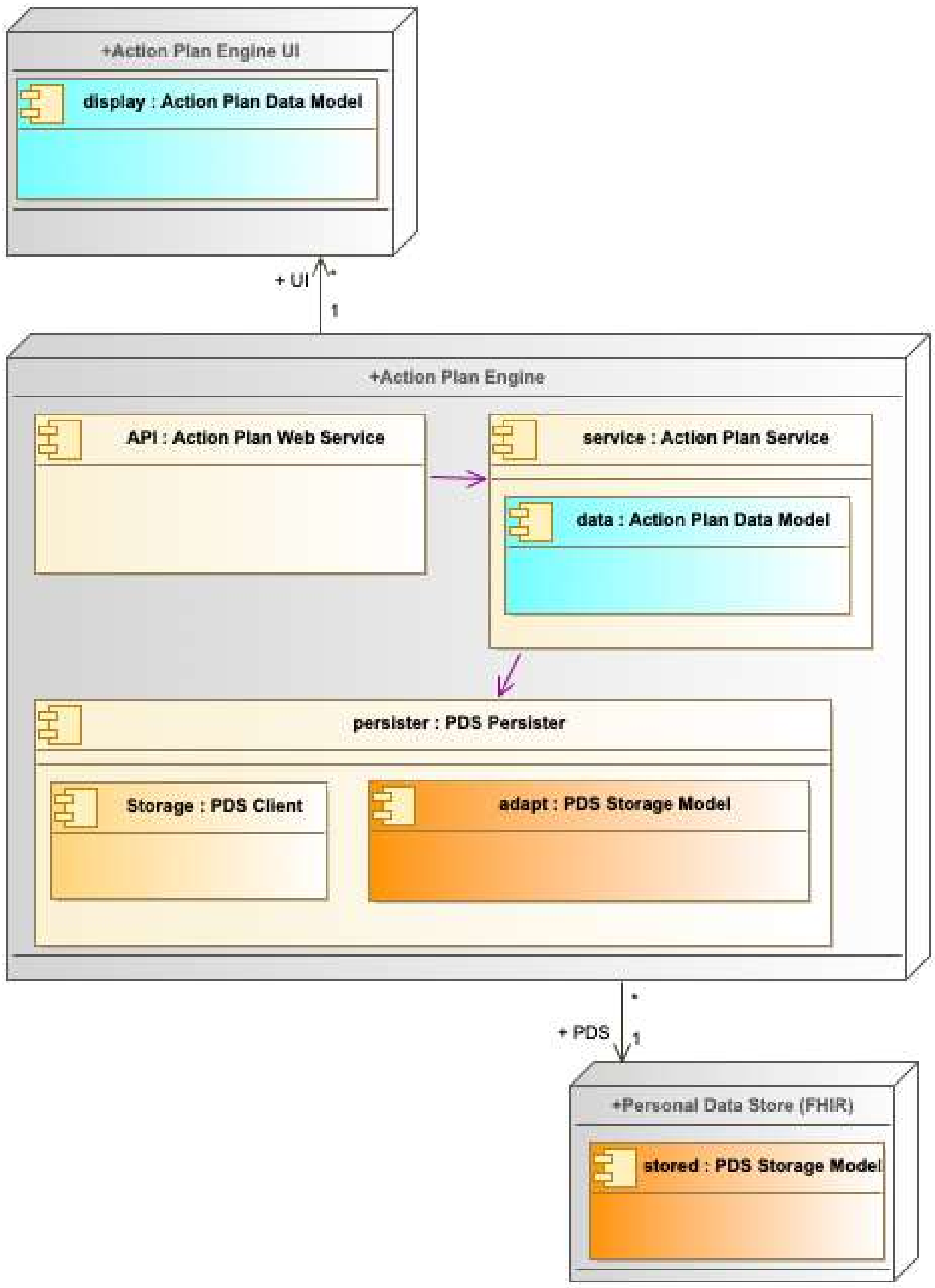

+Action Plan Engine UI
display : Action Plan Data Model
+ UI
*
1
+Action Plan Engine
API : Action Plan Web Service
service : Action Plan Service
data : Action Plan Data Model
persister : PDS Persister
Storage : PDS Client
adapt : PDS Storage Model
*
+ PDS
1
+Personal Data Store (FHIR)
stored : PDS Storage Model

Figure 9: Building Blocks of Action Plan Engine

As shown in Figure 9, there is one module named `action-plan-service` and several modules named `action-plan-service-xxx`. Each of the modules holds one implementation to connect the Action Plan Engine to a provided and configured storage backend. Beside the backend modules, other modules contribute to the action plan engine. Depending on the project configuration, the modules become part of the final deployment.

Table 1 gives an overview of the project's (sub) modules and their usage in the different usage scenarios.

Module	**Description**
action-plan-service	Core Module of the Action Plan Engine. This module provides the Web Interface layer and selects/uses the configured storage modules.
action-plan-service-pds	Provides the Connectivity to the Personal Data Store. This is the default storage module for the POWER2DM SMSS. This is the only persistence layer to be used in the POWER2DM Pilot Studies.
common-utils	As the name implies, helpful methods to be used throughout the Action Plan Engine
identity-provider	Provides information of the logged user and provides the (secure) user specific secure identifier used for storing and retrieving patient health records. See section 0 for details.
view-models	Provide the exchange format for Patient Health records used by the Action Plan Engine GUI (psmss-ui).
psmss-ui	Keeps the HTML/JS Files for the Action Plan Engine Web Interface. Can be hosted on a separate web server, or deployed together with the Action Plan Engine Web Application onto a servlet container.
psmss-webapp	Runtime project for the Action Plan Engine. Provides the runtime environment and configuration that is finally deployed in a Servlet Container.

Table 1: Action Plan Engine Module Overview

3.2 Storage Backend Selection

The distinct backend storage modules are selected based on a configuration value. Based on this configuration value, the respective backend storage endpoint is injected and used. Figure 10 outlines the following process:

1. The Action Plan Engine Webservice receives a request, e.g. for example to display the upcoming activities.
2. The (injected) Action Plan Service uses a `ServiceFactory` to obtain the requested service implementation.
3. From a list of all available service implementations, the desired implementation is selected.
4. The desired API method is invoked in order to retrieve the data properly formatted for the Action Plan Engine UI. For this, the storage component for reading/writing activities from/to the PDS is used to retrieve the data from the PDS.
5. Transparently, the adapter capable of adapting `ProcedureRequest` records from the PDS into Action Plan Engine's Activities (see Section 3.4) is used to transform the data retrieved from the PDS.

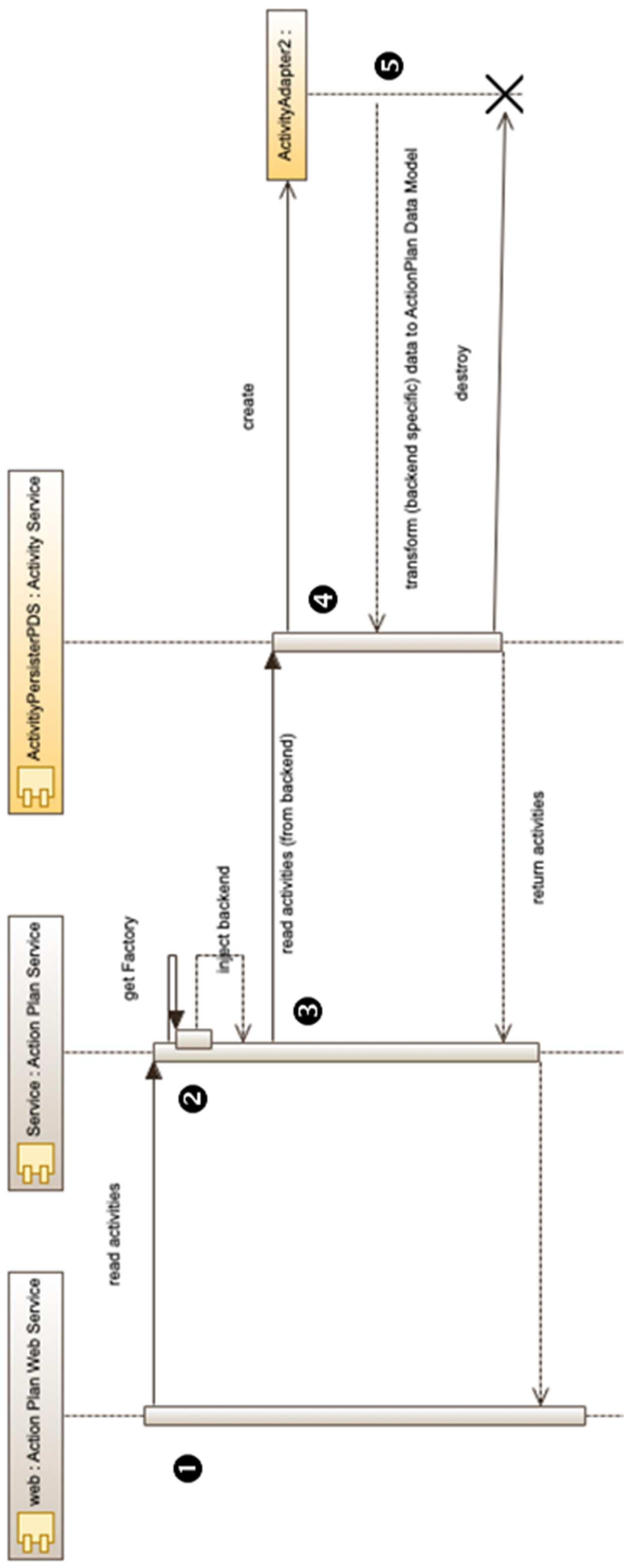

web : Action Plan Web Service
Service : Action Plan Service
ActivityPersisterPDS : Activity Service
ActivityAdapter2 :
1
read activities
2
get Factory
inject backend
3
read activities (from backend)
return activities
4
create
transform (backend specific) data to ActionPlan Data Model
destroy
5

Figure 10: Storage Endpoint Selection Example

This approach allows adding new storage endpoints at any time in separate project modules, without the need to change the existing modules. A similar approach is used for the identity management described in the next section.

3.3 Security

Patient Health Records (PHRs) represent very sensible, personal data. As a result, it is common to not store any of the PHR (glucose, weight measurements) along with identifying data such as name, birthdate or user credentials. Instead, any of the PHR is identified with a secure identifier representing the owner of the PHR. This particular identifier must not be provided to the user and his/her browser; instead it must be maintained in a trusted identity management service holding the user credentials and other personal information.

The Action Plan Engine however requires this secure identifier upon any request in order to identify the user's PHRs. To achieve this, any of the requests from the Action Plan Engine Web GUI must be accompanied with the current user's secure identifier, which in turn must be obtained from the trusted identity management service. Only in case the user is logged in and the identity management provides his/her secure identifier, the subsequent access to the backend storage components is possible.

The corresponding process is outlined in Figure 11.

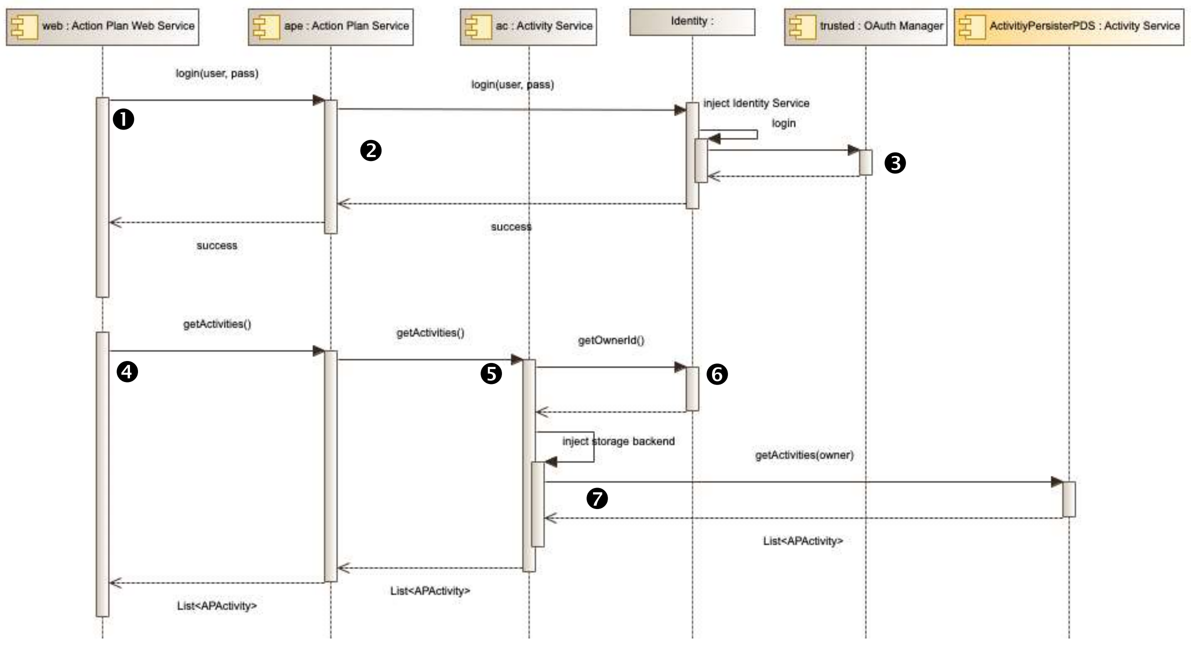


Figure 11: Obtaining Logged User's secure identifier

Whenever a user is not yet logged in, the Action Plan Engine GUI redirects to a login window from the Authentication Manager, where the user can provide his/her login credentials (❶). After successful login, a token is passed along to the Action Plan Engine (❷) to be used for accessing private resurces in the PDS. Internally, the identity management component is responsible to manage the logged users. This component has to select the currently configured trusted identity service (❸) and to verify the credentials. Such a trusted identity service can be an "Open ID Connect" endpoint as described in D4.9. This mechanism also allows integrating the services of the Action Plan Engine into a Single Sign-on environment.

Upon successful login, the user may access his/her data. The secure identifier must not be given to the Action Plan Engine GUI for subsequent data access; instead it is retrieved from the current user session. Figure 11 outlines this process in step (❹) where the Action Plan Engine GUI issues a request to the Action Plan Service, in this particular case to obtain the patient's activities. This request is handled by first delegating the request to the proper API service implementation (❺), hence obtaining the required secure identifier from the user's session (❻) and finally selecting the configured storage backend and retrieving the requested data based on the secure identifier (❼).

As a result of this approach, the secure identifier pointing to the patient's records is kept on the server. Furthermore, the patients' health records remain separated from personal information stored in a trusted environment.

3.4 Personal Data Store (PDS) Integration

In addition to internal data storage, the Action Plan Engine (APE) is also able to connect to external storages, such as the Personal Data Store (PDS) developed within POWER2DM. The PDS is used to store all patient related data and has been described in Deliverable D4.1 (Data Model) and D4.2 (Service API). Connectors and adapters from the APE to the PDS are described in this subsection.

The PDS Health data model contains the schema of all datatypes that can be stored in the PDS (D4.1). To facilitate the PDS storage within the Action Plan Engine, a library providing simple access methods for the PDS has been developed. This library is shown in Figure 12. The methods provided support the following access methods:

- reading of single PDS Data Objects, based on their ID
- searching of multiple PDS Data Objects by providing a basic type and a set of criterions
- writing of single PDS Data Objects supporting both creation of new objects and updating of existing data and
- deleting of single PDS Data Objects, based on their ID

In order to simplify the transformation of PDS Data Objects into their representation within the Action Plan Engine, the concept of the `PdsAdapter` has been introduced. A distinct `PdsAdapter` provides the required logic to transform a PDS Object of a given type into its Action Plan Engine representation. The PDS library gets all adapters registered on startup and automatically chooses the adapter to use depending on the processed objects.

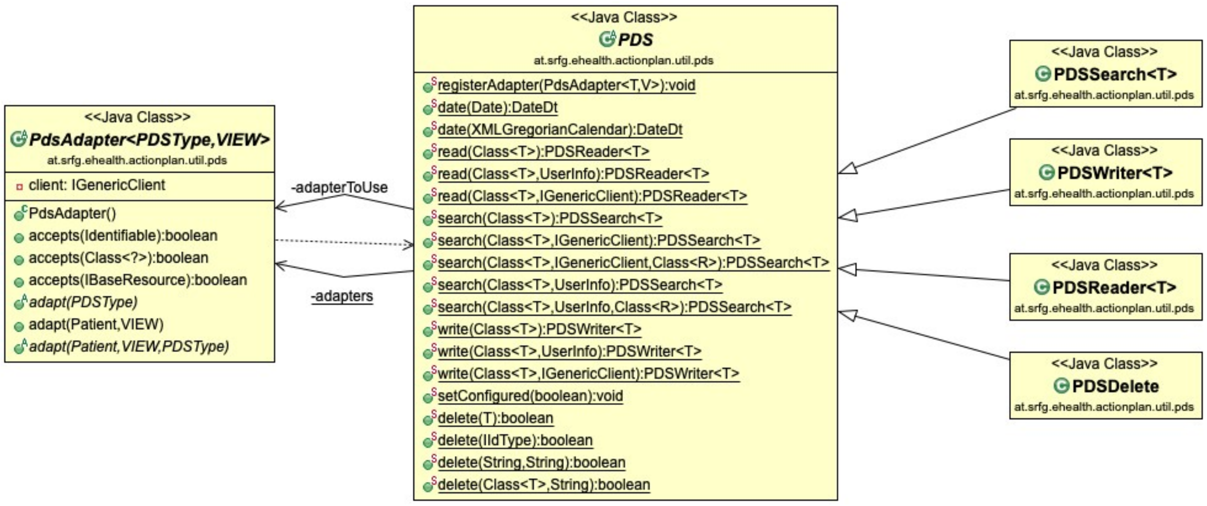

<<Java Class>>
PdsAdapter<PDSType,VIEW>
at.srfg.ehealth.actionplan.util.pds
client: IGenericClient
PdsAdapter()
accepts(Identifiable):boolean
accepts(Class<?>):boolean
accepts(IBaseResource):boolean
adapt(PDSType)
adapt(Patient,VIEW)
adapt(Patient,VIEW,PDSType)
-adapterToUse
-adapters
<<Java Class>>
PDS
at.srfg.ehealth.actionplan.util.pds
registerAdapter(PdsAdapter<T,V>):void
date(Date):DateDt
date(XMLGregorianCalendar):DateDt
read(Class<T>):PDSReader<T>
read(Class<T>,UserInfo):PDSReader<T>
read(Class<T>,IGenericClient):PDSReader<T>
search(Class<T>):PDSSearch<T>
search(Class<T>,IGenericClient):PDSSearch<T>
search(Class<T>,IGenericClient,Class<R>):PDSSearch<T>
search(Class<T>,UserInfo):PDSSearch<T>
search(Class<T>,UserInfo,Class<R>):PDSSearch<T>
write(Class<T>):PDSWriter<T>
write(Class<T>,UserInfo):PDSWriter<T>
write(Class<T>,IGenericClient):PDSWriter<T>
setConfigured(boolean):void
delete(T):boolean
delete(IIdType):boolean
delete(String,String):boolean
delete(Class<T>,String):boolean
<<Java Class>>
PDSSearch<T>
at.srfg.ehealth.actionplan.util.pds
<<Java Class>>
PDSWriter<T>
at.srfg.ehealth.actionplan.util.pds
<<Java Class>>
PDSReader<T>
at.srfg.ehealth.actionplan.util.pds
<<Java Class>>
PDSDelete
at.srfg.ehealth.actionplan.util.pds

Figure 12: PDS Library supporting reading (searching), writing, deleting and transforming of PDS data

The major datatypes that need to be handled by the Action Plan Engine are (as of writing this document):

- Observations and Clinical Results
- Goals (Treatment Goals and Self-Management Goals)
- Action Plans (Treatment Plans and Self-Management Plans)
- Medication Orders and Administration
- Questionnaires & Questionnaire Results
- User Settings

- User Profile Information

These distinct data types are part of the Action Plan Engine Data Model (see section 2.3) and are also used in the Action Plan Engine's PDS storage component by providing the respective `PdsAdapter` implementations as shown in Figure 13. The storage comp

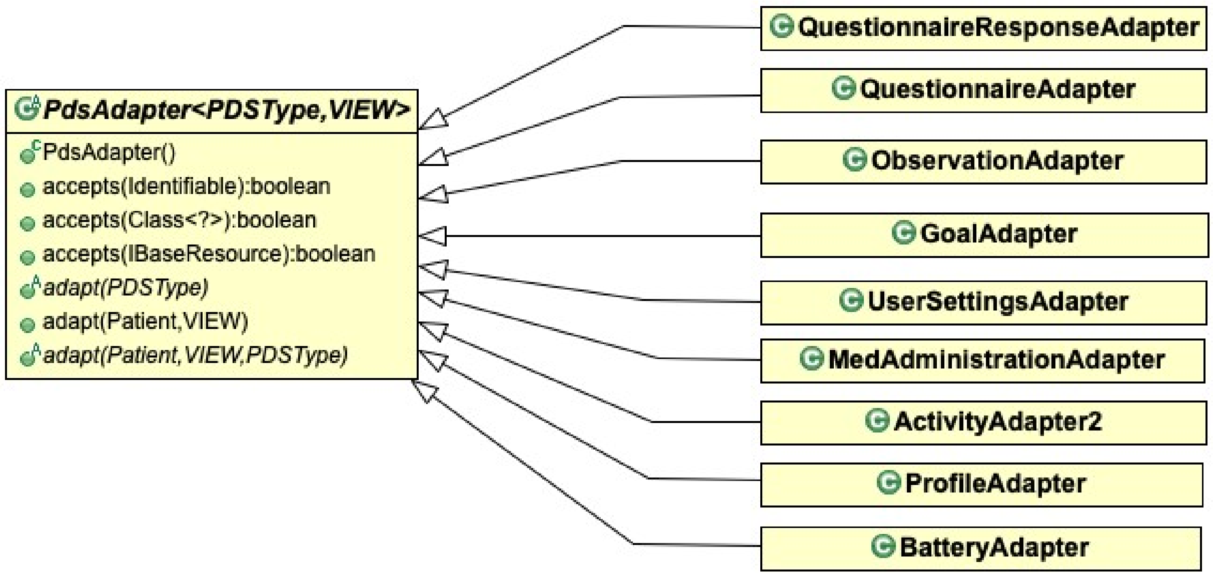


Figure 13: PdsAdapter implementations

The Action Plan Engine uses the distinct adapters by querying the PDS transparently. The adapters are chosen depending on the requested PDS data type and its corresponding Action Plan Engine Data Model element.

3.5 REST Web Service API

The REST API provided by the Action Plan Engine is mainly used by the Action Plan GUI as described in the next subchapter (Section 3.6). In addition, the mobile app accesses the API for retrieving upcoming tasks and short reviews on the performance achieved in the previous periods. In the following we describe the API services available. The full documentation of the API is made available to all developers using "Swagger"[7], a powerful tool to deliver OpenAPI Specifications for REST services.

Services of the actionplan engine, version 2:

```
get    /v2/actionplan/activity
post   /v2/actionplan/activity
delete /v2/actionplan/activity
get    /v2/actionplan/activities

get    /v2/actionplan/goal
post   /v2/actionplan/goal
delete /v2/actionplan/goal
get    /v2/actionplan/goals
```

[7] https://swagger.io/

```
get    /v2/actionplan/observation
post   /v2/actionplan/observation
delete /v2/actionplan/observation
get    /v2/actionplan/observations

get    /v2/actionplan/settings
post   /v2/actionplan/settings

get    /v2/actionplan/calendar
get    /v2/actionplan/review
get    /v2/actionplan/tasks
get    /v2/actionplan/terms
get    /v2/actionplan/treatmentgoals
get    /v2/actionplan/{ownerId}/review
get    /v2/actionplan/{ownerId}/reviews
get    /v2/actionplan/{ownerId}/tasks
```

Services for managing questionnaires:

```
get    /v2/questionnaire
get    /v2/questionnaire/list
post   /v2/questionnaire
```

3.6 GUI Overview

The following chapters are going to provide a brief overview of the different GUI components and technical approaches. A more in-depth description for the user point of view is provided in Deliverable D3.8 – "Web-Based GUI Components for DSS".

3.6.1 GUI Components

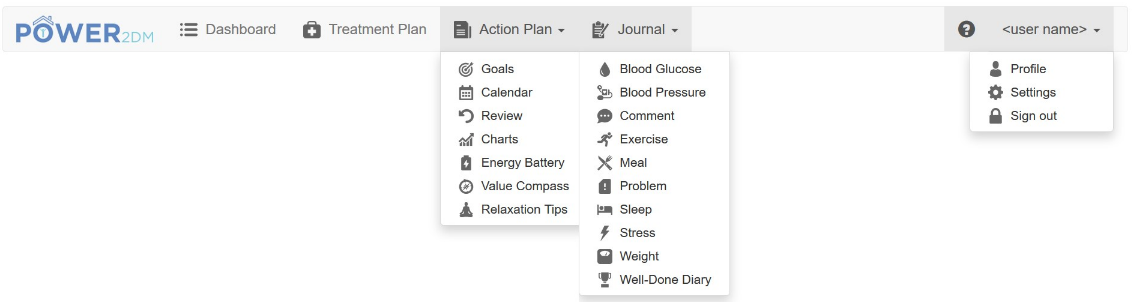


Figure 14: Main navigation

The GUI is composed of the following elements:

- Dashboard (reachable by clicking on "POWER2DM" " or "Dashboard"): Overview page of the most relevant data (such as performance of past week, upcoming activities, charts). Presented elements can be customized by the user in their settings.
- Treatment Plan: Lists relevant goals and activities ordered by a practitioner. This list is read-only but goals and activities can be imported to the Action Plan.
- Action Plan: Handles all components related to self-management.
  - Goals: Lists relevant self-management goals (Self-management Goals can be referenced to Treatment Goals).
  - Calendar: Presents self-management activities in an adjustable calendar view (Activities can be referenced to Self-management Goals).

  - Review: Shows a performance summary for the past week (or month, quarter, etc.). Detailed information is given in Section 3.8, “Review and Interventions”
  - Charts: Presents recorded patient ODL data in chart views with an adaptable timeframe.
  - Energy Battery: Provides a guided workflow on how to regain energy in case of low energy/mood. Detailed information is given in Section 3.9 “Energy Battery”.
  - Value Compass: Provides a guided workflow on how to identify personal values and possible barriers. Detailed information is given in Section 3.10 “Value Compass”.
  - Relaxation Tips: Lists customizable tips presented to the user in case of stress situations.
- Journal: Lists recorded patient ODL data in various categories.
- Profile and Settings (reachable by clicking on the user name drop-down): Collects all patient-related (Profile) and display-related (Settings) data.

Detailed Mock-ups and Screenshots for each menu item are available in D3.7.

3.6.2 Single-page Layout

The web application pursues a single-page layout by loading content dynamically using various methods described in this section. In general, all globally relevant HTML and JS content is added to the central index.html file while the corresponding content for each view is retrieved on runtime.
In order to retain the browser native back/forward/reload methods and allow for deep linking the URL hash notation is used (e.g. https://someurl.com/#somepage). The hashes can be broken down further for additional functional or navigational purposes using underscores (e.g. https://someurl.com/#someaddition_somepage).

Whenever the URL hash changes (**onhashchange** event), the function **init()** is called. It splits the current location href into base and hash and passes the hash on for further processing.
The function **show(page,id)** handles loading the relevant resources into the DOM. It splits the hash by underscores (in case there are any), loads the HTML file for the requested page and passes on the page and additional underscore attributes for further processing. By default, the HTML file is loaded into the main content DIV but the id parameter allows for loading content into any DIV – this enables the developer to assemble a page using any amount and layout of components without code duplication.

The function **performAction(page,add)** takes care of executing onload methods (e.g. loading data via AJAX) and processing the additional underscore attributes.

3.6.3 Dynamic Content Generation

Some content and layout types are very congruent throughout HTML pages (e.g. input forms). In order to keep code duplication and complexity to a minimum, JSON templates are used for definitions that are translated to HTML using JS on runtime.
For example, all ODL tables and input forms are generated from the following template format:

```
{
  "ODLTYPE": {
    "label": STRING,
    "ATTRIBUTE": {
      "chart": BOOL,
      "dataType": STRING,
      "display": STRING,
      "label": STRING,
      "outer": STRING,
      "type": STRING,
      "unit": STRING,
      "values": [STRING]
    }
  }
}
```

Each ODLTYPE has a label (an internationalization tag as a STRING) for display purposes and various ATTRIBUTEs. Those contain the following keys:

- chart: BOOL indicating whether or not the attribute should be displayed in the charts
- dataType: The datatype of the attribute (e.g. "STRING", "NUMBER", "ISO_DATE", etc.)
- display: STRING indicating if and where to display the attribute
- label: Internationalization tag as a STRING for display purposes (attribute name)
- outer: Potential outer JSON key as STRING when retrieving/sending the attribute
- type: STRING indicating the attribute type (e.g. "REQUIRED", "OPTIONAL", "GENERATED", …)
- unit: Internationalization tag as a STRING for display purposes (attribute unit name)
- values: ARRAY of internationalization tags as STRINGs in case of predefined answers for multiple choice or likert scale dataTypes

3.7 Decision Trees

3.7.1 Decision Tree Modelling

A separate tool has been developed using the following open-source JS libraries:

- Bootstrap: Responsive design framework (MIT license)
  http://getbootstrap.com/
- jQuery: Multi-feature framework (MIT license)
  https://jquery.com/
- visJS: Visualization framework (Apache 2.0 and MIT license)
  http://visjs.org/

The general purpose is to enable non-technical partners (e.g. practitioners) to define decision tree workflows in a simple yet structured way that allows for a consistent and fast implementation on the technical level. An example decision tree for problem identification is shown in Figure 15.

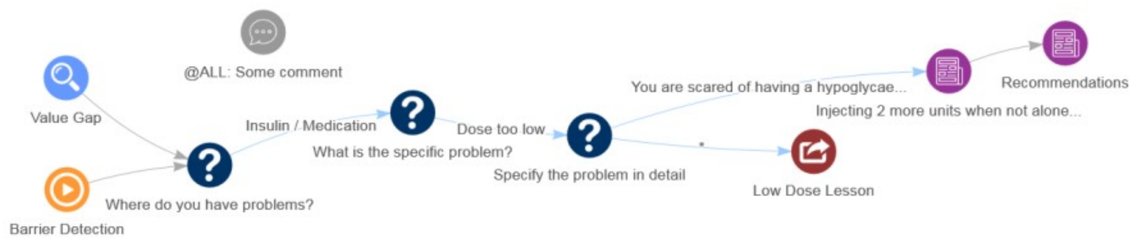

@ALL: Some comment
Value Gap
Barrier Detection
Where do you have problems?
Insulin / Medication
What is the specific problem?
Dose too low
Specify the problem in detail
You are scared of having a hypoglycae...
Injecting 2 more units when not alone...
Recommendations
Low Dose Lesson

Figure 15: Example decision tree workflow

There are various node types available:

- Action: Manually activate specific triggers
- Comment: Add comments when collaborating on a workflow
- Content: Show recommendations, specific pages, static content, etc. to the user
- Condition: Check specific conditions
- Question: Any type of question

- Trigger: Manually or periodically trigger a (sub-)workflow

Furthermore, there are two edge types:

- Answer: Forward to the next node if the response data of the preceding question matches
- Forward: Forward after the preceding node has finished

A complete manual on the usage of the current version of the tool and associated notations can be found in the Appendix. The manual is also delivered as downloadable version from the GUI of the workflow tool itself, which is delivered with each deployment of the action plan engine.

3.7.2 Decision Tree Execution

The decision tree execution process is based on JSON templates that are generated from visJS JSON exports and translated to the following format:

```
{
  "DECISIONTREE": {
    "txt": STRING,
    "i18n_txt": STRING,
    "choices": [
      {
        "txt": STRING,
        "i18n_txt": STRING,
        "action": {
          "type": STRING
          "ref": STRING
        }
      }
    ]
  }
}
```

Each DECISIONTREE has a txt/i18n_txt as a STRING that is displayed and various choices. Those contain the following keys:

- txt/i18n_txt: Human readable text and internationalization tag as a STRING of the node / question text
- action: JSON element with the following keys:
    - txt/i18n_txt: Human readable text and internationalization tag as STRING of the action text on leaf nodes
    - type: Action type as a STRING of the following DECISIONTREE (e.g. subtree, intervention, …)
    - ref: STRING to reference the target DECISIONTREE or intervention

This JSON structure and associated functionality is directly interpreted on the front-end. Automated export methods to the open-source tool Node-RED[8] (Apache 2.0 license) have been prepared for easy back-end integration and more advanced features.

[8] https://nodered.org/

3.8 Review and Interventions

This prototypes' review component supports the periodic review cycle that the patient is encouraged to perform at least on a weekly basis. By default the last seven days are assessed, although the patient can modify the assessment time over longer periods.

The Action Plan review does following Kate Lorig's[9] model for patient self-management, providing an action plan to record an action plan that enables a patient to plan and thereby schedule an activity such as glucose monitoring. The adherence performance scoring mainly aims to support behavior change and provide intervention tips to help them adhere to an activity plan to support his/her goals designed by the patient following treatment recommendations.

3.8.1 Performance Assessment

For the current prototype, only adherence or compliance performance is considered. The patient might adhere to the activity plan she first made and monitor herself by recording (or automatically) by posting activity ODLs or acknowledgements that she performed the planned activity. However, so as not demotivate anyone, simply posting ODLs is also fine. For the second prototype, additional user settings should be provided to enable the patient to define their performance assessment rules.

The performance is based on the adherence to the planned activity for the given time interval selected by the user. A score is calculated to not only give feedback to the patient, but also trigger an appropriate tip or textual intervention. The text tip can include additional links to either activate a decision tree enabled intervention or provide supplemental external infos (videos, patient info, and additional tips).

The current implementation focuses on supporting behavior change. Instead of providing "judgmental" assessment of goals based on measurements e.g. Body weight goals, the behavior change focuses on the assessment of performing small efforts, e.g. activities, that will slowly lead to the achievement of particular goals. Activities are scheduled to support the building of habits. Scoring of activity performance is based on adherence or compliance to a schedule plan. For particular ODL activities, scoring might require adherence to a strict time e.g. eat a snack at about 10:00 with a narrow time window (+/- 30 minutes) versus go jogging at 18:00 with a wider time window (+/- 3 hours). In the future, users should set their own scoring rules. However, even if the patient chooses not to create an activity plan, they are scored positively when they at least record their activities.

The adherence performance of each goal, consequently, is based on one or more activities. A goal's associated activities are assigned when a new activity is created; the patient is asked to assign one or more previously defined goals. Furthermore, the activity can be updated anytime to update the assigned goals.

3.8.2 Implementation Overview

3.8.2.1 Review Services

Review Services build a review object that can be shared with various components, including the GUI or the mobile application. The review object is created by adding recorded events. Any event might be attached to a particular activity type and consequently a particular activity. Subsequently any activity

[9] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1497631/

may have related goals which in turn may point to related treatment goals. These relationships are finally represented as recursive review sub-categories where each category computes its own review score. The categories computed are as follows:

Category	Collected Events	Sub-Categories
Overall	All Events added to the review are collected in this category.	• Activity Types: Whenever an event is assigned to an activity a new sub-category computing the overall score for all activity types is appended. • Goals: Whenever an event's activity is related to a goal, the overall score for goals is appended. • Treatment Goals: Whenever an event's activity's related goal points to a treatment goal, the overall score for treatment goals is appended.
Activity Types (Overall)	All events recorded for any Activity Type are collected in this category.	• Activity Type: The distinct activity type (Sleep, Exercise, Meal etc.) is appended as sub-category.
Activity Type (Sleep, Stress, Meal, Exercise, etc.)	All events recorded for a particular activity of the given Activity Type are recorded in this category	• Activity: A sub-category identified by the planned activity is added to the review object.
Activity	All events recorded for a distinct activity are collected	• No sub-categories
Goals (Overall)	All events whose activity is related to any goal.	• Goal: A sub-category identified by the goal is added to the review object
Goal	All events whose activity is related to the category's goal are collected.	• No sub-categories
Treatment Goals (Overall)	All events whose activity is related to a goal which in turn has related treatment goals	• Treatment Goal: A sub-category identified by the treatment goal is added to the review object
Treatment Goal	All events where one of the activity's goals relates to the category's treatment goal.	• No sub-categories

Table 2: Review Category Hierarchy

Any of the review categories maintains its own score object, thus collecting the number of planned, completed and observed events. Based on the computed scores, the review object may receive additional tips out of the intervention table by injecting a tips-processor. The tips-processor applies several rules to identify appropriate interventions based on the review category and the computed score. The full structure of the review objects is outlined in Figure 16.

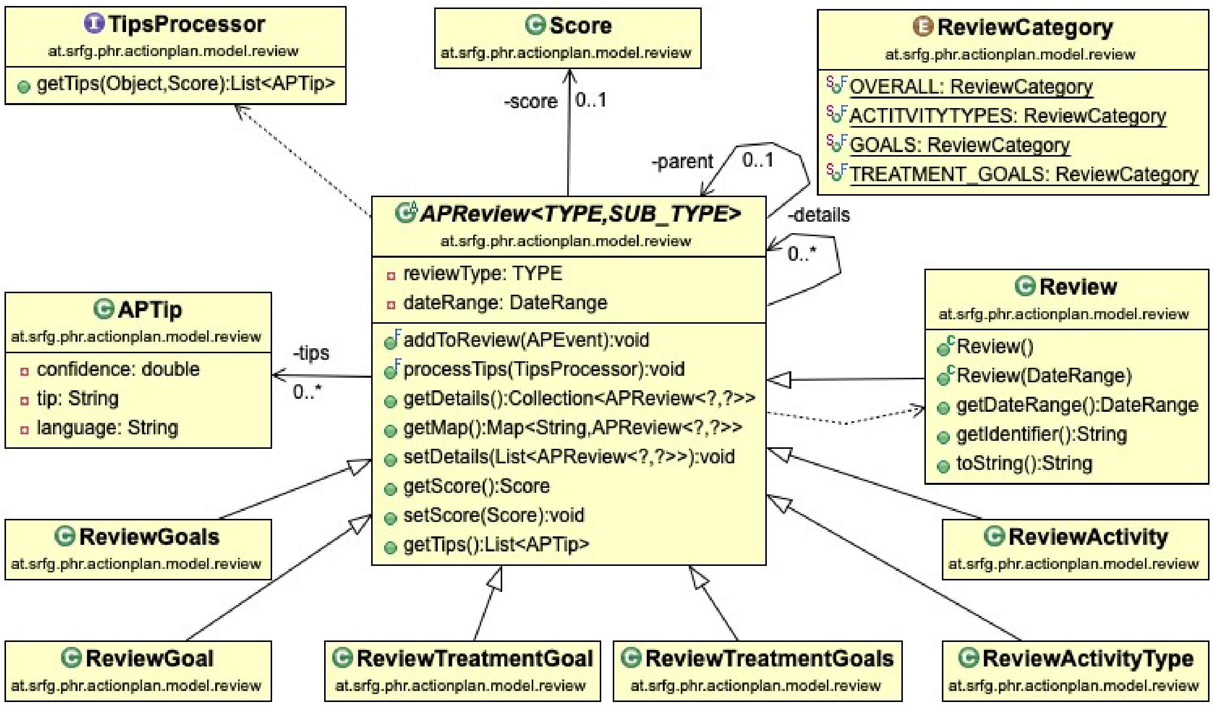

TipsProcessor
at.srfg.phr.actionplan.model.review
getTips(Object,Score):List<APTip>
Score
at.srfg.phr.actionplan.model.review
ReviewCategory
at.srfg.phr.actionplan.model.review
OVERALL: ReviewCategory
ACTITVITYTYPES: ReviewCategory
GOALS: ReviewCategory
TREATMENT_GOALS: ReviewCategory
-score
0..1
-parent
0..1
-details
0..*
APReview<TYPE,SUB_TYPE>
at.srfg.phr.actionplan.model.review
reviewType: TYPE
dateRange: DateRange
addToReview(APEvent):void
processTips(TipsProcessor):void
getDetails():Collection<APReview<?,?>>
getMap():Map<String,APReview<?,?>>
setDetails(List<APReview<?,?>>):void
getScore():Score
setScore(Score):void
getTips():List<APTip>
APTip
at.srfg.phr.actionplan.model.review
confidence: double
tip: String
language: String
-tips
0..*
Review
at.srfg.phr.actionplan.model.review
Review()
Review(DateRange)
getDateRange():DateRange
getIdentifier():String
toString():String
ReviewGoals
at.srfg.phr.actionplan.model.review
ReviewActivity
at.srfg.phr.actionplan.model.review
ReviewGoal
at.srfg.phr.actionplan.model.review
ReviewTreatmentGoal
at.srfg.phr.actionplan.model.review
ReviewTreatmentGoals
at.srfg.phr.actionplan.model.review
ReviewActivityType
at.srfg.phr.actionplan.model.review

Figure 16: Review Implementation

Upon review processing, the planned activities (e.g. "Measure blood glucose every day at 8:00am") are retrieved from the PDS. For each of the planned activities, the corresponding (planned) events are created and then matched with the recorded observations. Whenever a recorded observation can be assigned to a planned event (e.g. "Measure blood glucose on Monday, 22nd of October at 8:00am), this planned event is marked as completed and added to the review. Events where no matching observation can be found are added as planned to the review. Finally, the observations, where no planned activity is present are also added as observed events to the review. Whenever an event is added to the review, the score of the overall review object is computed. Depending on the existence of related activities, goals or treatment goals, the corresponding sub-category's score is updated accordingly. The score finally provides the number of planned, completed and additionally observed events for every review category. The category and the computed score for each (sub-)category are then used to identify interventions to be shown to the patient.

The matching algorithm between planned events and recorded activites is different based on the type of activity. While glucose, meal and and blood pressure measurements match only within 2 hours according to the plan, other activites such as physical activities are also counted as fulfilled when they are done within the same day.

3.8.2.2 Intervention Table - Intervention Motivational Tips

The intervention table was created by domain experts and rules were derived from the table. The tip text format supports a lightweight markup language called Markdown[10] so there is no dependency on HTML for URLs, or lists formatting. Table 3 outlines the interventions text along with the unique identifier for each intervention.

[10] https://en.wikipedia.org/wiki/Markdown

Text ID	Intervention Text
004	You are making progress on your goal. Good! There might also be things that you would still like to improve. Could you think about what could you do different next week? Try to plan one specific thing for the next week, of which you think it may even further improve your progress. You can plan it on your Power2DM calendar/agenda.
009	"Well done! You are doing a great job"
010	Keep up the good work! Hopefully the progress on your goal makes you feel more positive about yourself. Try to reflect on positive feelings about making progress on this goal.
011	You are doing well! Try to consciously think about how it makes you feel to achieve your goal. Do you feel proud, confident, happy, cheerful, excited? Or is there another positive emotion that you recognize? Think about it, and try to write down all the positive feelings that you have. Even the "smallest" positive feelings are important to recognize. You can use our [well-done diary](*#well-done-diary)*
012	Do you feel adequately informed about diabetes and its effects on you? If yes, that is important and really good! If not, you might take a look the *[following information material](http://www.diabetes.co.uk/emotions/)*. Many people with diabetes underestimate the effect it has on them, or feel that they should not complain about it.
013	Good that you are still trying! Continue your good work!
014	Keep going! It might help to consider in which situations you have already succeeded in achieving your goals
015	Good that you are still trying! Continue your good work!
016	Have you made your goal sufficiently clear and realistic so that it can reach it with reasonable effort? If not, try to reformulate your goal and your activity planning and put them into practice.

Table 3 – Intervention Table sample

Table 3 outlines solely the textual part of the intervention table. In addtion, every intervention is available in different languages which are omitted. The provided interventions may utilize Markdown especially to represent hyperlinks or lists e.g. *[label or text](hyperlink*). An example can be found in Table 3, TextId "012" linking the user to external information material.
When selected, the provided intervention is rendered as follows:

Do you feel adequately informed about diabetes and its effects on you? If yes, that is important and really good! If not, you might take a look the following information material. Many people with diabetes underestimate the effect it has on them, or feel that they should not complain about it.

Hyperlinks containing only an anchor symbol ("#") are internal links, especially to enable the user to start a particular decision tree process in the decision tree GUI or to begin a diary, etc. e.g. at row 011:

You can use our [well-done diary](#well-done-diary)

The complete intervention table is part of D3.2 ("Dynamic Behavior Change Intervention Models for Self-Management II"). Finally, the intervention table provides additional conditions/rules to apply before providing the intervention to the patient. This particular processing of the interventions is explained in detail in section 3.8.2.3 below.

3.8.2.3 Tips Processing

As shown in Figure 16, the basic review object (`APReview`) allows processing the tips by providing/injecting a `TipsProcessor`. This processor is called for every review category along with the computed score. Along with the textual tips to the patient, the intervention table provides a number of constraints defining when an intervention is eligible to be shown to the patient.

Text Id	Category	ODL Type	Conditions/Risks	Compliance
004	Overall Goals Activities			Performance == 4
009	Overall Goals Activities		Male	Performance == 3
013	Goals Activities	Exercise		Performance < 3
014	Goals Activities	Glucose	Smoker	Performance < 3

Table 4: Intervention constraints

The following steps outline the general steps when selecting tips any category. For each of the review categories, the following constraints are evaluated step by step where each step eliminates interventions are not appropriate in the given situation.

1. Filter Review Category: All interventions with the requested category are selected in this step
2. Filter ODL Type: When set, only activities/goals of the requested type are selected in this step
    a. Optional: Filter for ODL-Sub-Types such "Exercise – Walking"
3. Filter Patient Profile: When set, check for gender and other personal settings.
4. Filter Performance/Adherence: Check the computed score whether it matches the performance constraint. The score performance is computed based on the planned, completed and observed numbers.

After checking all the distinct rules, at least one of the valid interventions is selected randomly and added to the resulting review object.

3.9 Energy Battery

This chapter only provides a brief overview of the technical approaches used during the implementation of the Energy Battery. A more in-depth view of the background and workflow is provided in Deliverable D3.8 – "Web-Based GUI Components for DSS".

The workflow consists of three steps:

- Raise awareness: A short text and video introduction on how energy is lost and gained
- Energy Diary: A simplified calendar add-on for setting up schedules assigned to "energy categories" and a personal assessment on their extent
- Feedback: Textual and visual feedback on the Energy Diary input

The feedback itself is contructed from the average daily sleep duration and the personal assessment. The results of the Energy Battery are sent to the backend, using the datamodels as described in Section 2.3.4. In the PDS, the Energy Battery entries are stored as "ProcedureRequest", using a special ActivityType and FHIR HL7 status "proposed".

3.10 Value Compass

This chapter only provides a brief overview of the technical approaches used during the implementation of the Value Compass. A more in-depth view of the background and workflow is provided in Deliverable D3.8 – "Web-Based GUI Components for DSS".

The workflow is soley implemented in the front-end and consists of eight steps:

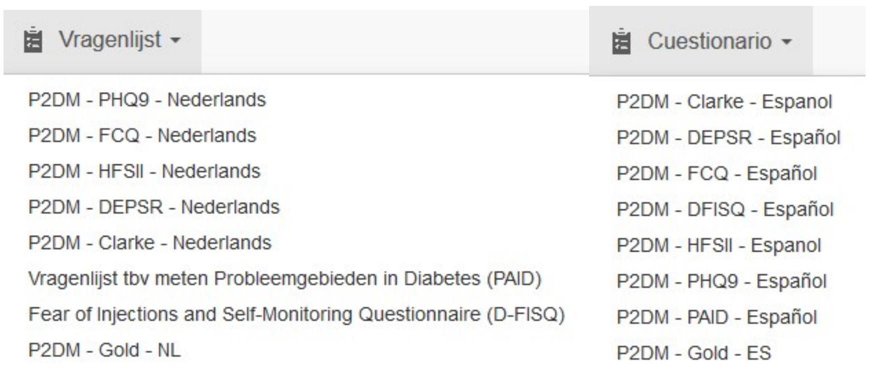
- Introduction: A short text introduction on the meaning of values
- Imagination: A textual scenario in order to prepare the user for the following tasks
- Life Areas: An overview of the different life areas that are going to be assessed
- Brainstorming: A more advanced description of the selected life area as well as input fields for notes and derived "key sentences"
- Importance: Sliders allowing for rating the importance of all derived "key sentences"
- Contentment: Slider allowing for rating the contentment of the most important "key sentences" identified in the previous step
- Gaining Clarity: Visual feedback on the Importance and Contentment data
- Gaps: Textual and tabular feedback on the Importance and Contentment data as well as predefined actions for identified "gaps"

The feedback itself is constructed from the Importance and Contentment ratings.

The result data produced by the value compass is stored to the PDS in the "UserSettings" object using the corresponding adapter as described in Section 3.4.

3.11 Questionnaires

The questionnaires are only technically implemented as part of the Action Plan Engine. They are not an integral part in the self-management process. Also, they only appear if the backend provides the structured information of question and answer options, which are only available for the pilot studies in Spanish and Dutch. If available a new menu item is integrated into the main navigation bar as shown in Figure 17.



Vragenlijst	Cuestionario
P2DM - PHQ9 - Nederlands	P2DM - Clarke - Espanol
P2DM - FCQ - Nederlands	P2DM - DEPSR - Español
P2DM - HFSII - Nederlands	P2DM - FCQ - Español
P2DM - DEPSR - Nederlands	P2DM - DFISQ - Español
P2DM - Clarke - Nederlands	P2DM - HFSII - Espanol
Vragenlijst tbv meten Probleemgebieden in Diabetes (PAID)	P2DM - PHQ9 - Español
Fear of Injections and Self-Monitoring Questionnaire (D-FISQ)	P2DM - PAID - Español
P2DM - Gold - NL	P2DM - Gold - ES

Figure 17: Questionnaires in Dutch and Spanish

Questionnaires in the pilot studies can only be answered within pre-defined schedules from the clinicians. Only at the beginning as well as in regular intervals during the runtime of the study, the questionnaire will be possible to be filled by the patient. Questionnaire results cannot be modified or deleted anymore by the patient once the answers are submitted and scores have been calculated by the backend.

4 Action Plan Cloud Deployment

The cloud environment for the acceptance testing and production use in the pilots is hosted by TNO and managed by PD. It requires the applications to be deployed within Docker[11] containers. Therefore the Action Plan Engine provides all releases as pre-packaged docker images, currently using the official Tomcat image version “tomcat:7-jre8” from docker hub as its base image.

4.1 Action Plan Release Management

All developments of the POWER2DM Action Plan Engine are hosted in a separate GIT repository at Salzburg Research, containing all the modules described above, organised in two major branches:

- Active development is done on the ‘*develop*’ branch. All changes are compiled and tested. If all tests are successfully finished, changes are directly deployed to the developer test environment described in Section 4.2.
- Releases are created from the ‘*master*’ branch. Only changes that have been tested on the developer test environment are merged into the master branch. All releases are individually tagged with a version tag. The version number consists of three numbers: A major version number, related to the “Prototype version 1, 2, 3 and 4”, a minor version number, always increased, if a major API change has been applied, and a bug-fix number, which is increased once small bug-fixes are required to be released immediately, before the next minor version number gets available. Each release is again tested and deployed to a test environment, before the deployment may be done to the acceptance environment and finally to the production environment.

In addition several submodules are integrated into the workflow:

- pds-client: Provides the access client classes to the PDS.
- pds-data-generator: Programs to generate test data for the PDS based on (large) text input files.
- pds-test-data (deprecated): Test classes to generate PDS test data on the fly.

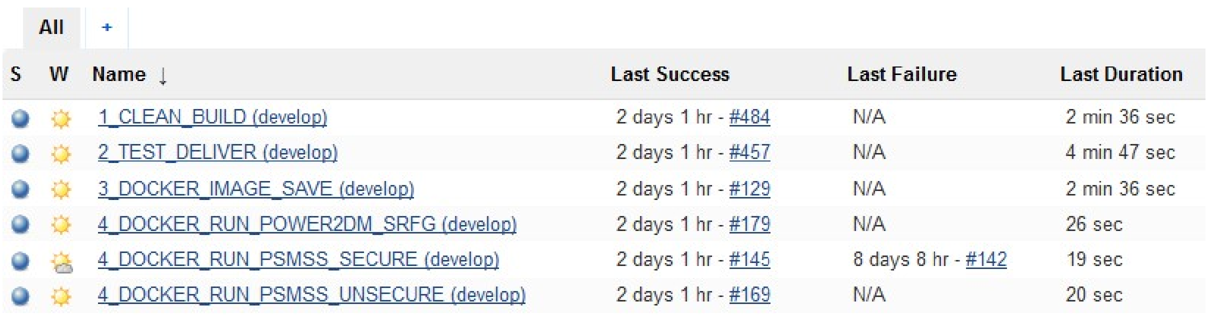
[11] https://www.docker.com

Clones of those repositories hosted at SRDC are available in the GIT repository at Salzburg Research.

4.2 Action Plan Delivery Process

The testing and release process of the Action Plan Engine is fully automated from the source code repository until the deployable Docker image repository located in the Netherlands. The process automation is carried on with the Jenkins Continuous Integration Framework on a local instance at Salzburg Research.

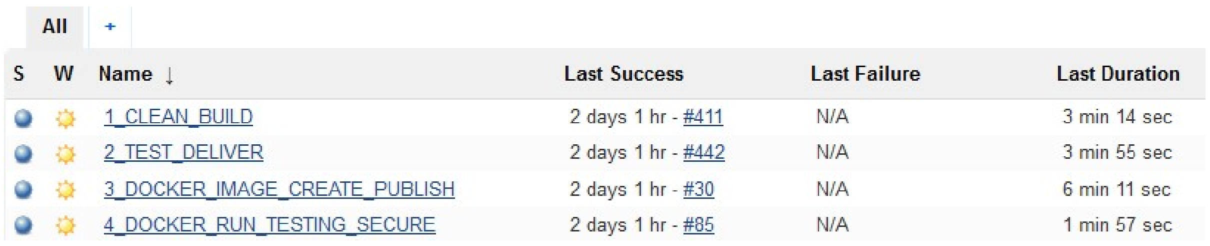
Also, additional test environments are deployed at the servers from Salzburg Research for local testing. Three deployents are done using the "develop" branch to be able to test recent developments using different backends (secure and unsecure version for easier testing), and one for the "master" branch, a secure and more stable environment including access to log files. The build and deployment pipelines from Jenkins are shown in the following figures.



All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	1_CLEAN_BUILD (develop)	2 days 1 hr - #484	N/A	2 min 36 sec
●	☀	2_TEST_DELIVER (develop)	2 days 1 hr - #457	N/A	4 min 47 sec
●	☀	3_DOCKER_IMAGE_SAVE (develop)	2 days 1 hr - #129	N/A	2 min 36 sec
●	☀	4_DOCKER_RUN_POWER2DM_SRFG (develop)	2 days 1 hr - #179	N/A	26 sec
●	☀	4_DOCKER_RUN_PSMSS_SECURE (develop)	2 days 1 hr - #145	8 days 8 hr - #142	19 sec
●	☀	4_DOCKER_RUN_PSMSS_UNSECURE (develop)	2 days 1 hr - #169	N/A	20 sec

Figure 18 Build and Deploy Pipeline for the "develop" branch



All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	1_CLEAN_BUILD	2 days 1 hr - #411	N/A	3 min 14 sec
●	☀	2_TEST_DELIVER	2 days 1 hr - #442	N/A	3 min 55 sec
●	☀	3_DOCKER_IMAGE_CREATE_PUBLISH	2 days 1 hr - #30	N/A	6 min 11 sec
●	☀	4_DOCKER_RUN_TESTING_SECURE	2 days 1 hr - #85	N/A	1 min 57 sec

Figure 19 Build and Deploy Pipeline for the "master" branch

The publishing of docker images to the POWER2DM docker repository is done only within the "master" branch, where the version number must be tagged correctly. Once the image is published it can be deployed for acceptance testing and pilot studies ("production").

5 Conclusion

This document outlines the current state of the implementation of the Action Plan Engine, which is deployed for the feasibility and pilot studies in Spain and the Netherlands as "Prototype 2". The work on the Action Plan Engine is ongoing, on the one hand for providing fixes for bugs that occur during the pilot studies, and on the other hand to have an updated "Prototype 3" for the upcoming pilot phase.

APPENDIX I – WORKFLOW TOOL MANUAL

1. Workflow Tool

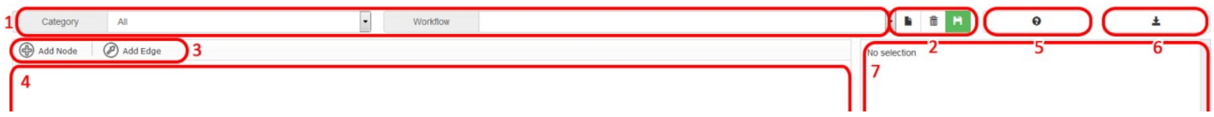


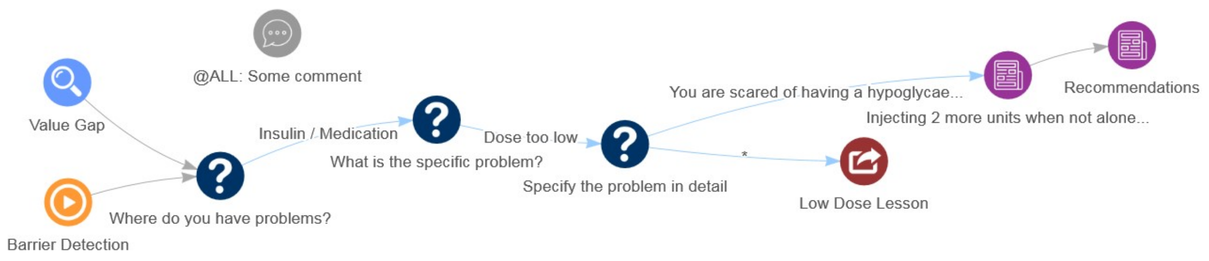
Figure 20 – Main Window

Use the category and workflow select box (see item 1 in Figure 20) to load an existing workflow.
The top-right buttons (see item 2 in Figure 20) can be used to create an empty workflow (left), delete the current workflow (center) or save/update the current workflow (right).
Important: In case you want to collaborate on the same workflow, it is advisable to always "Save as new workflow" instead of updating. Track the version number in the title for clarity and use the comment field for tracking the "Who? What? When?".
Delete actions always have to be confirmed in a separate pop-up and warnings are triggered before leaving the current workflow in case there are unsaved changes to the data.
The workflow interaction buttons (see item 3 in Figure 20) can be used to add, edit or delete nodes and edges. The buttons change according to the selected elements. Details can be found in "Nodes & Edges".
The drawing area (see item 4 in Figure 20) displays the current workflow (see Figure 21) and offers various interactions:

- The whole workflow can be zoomed using the scroll wheel
- The whole network can be moved by using drag&drop on the background
- Nodes can be arranged by using drag/drop on them
- Nodes and edges can be selected (for editing / deleting) by clicking on them

The information area (see item 7 in Figure 20) display the properties of a select node or edge.
The help icon (see item 5 in Figure 20) opens the Workflow Tool Manual.
The download icon (see item 6 in Figure 20) offers the functionality to download the current workflow as a PNG image (see **Download Workflow**).

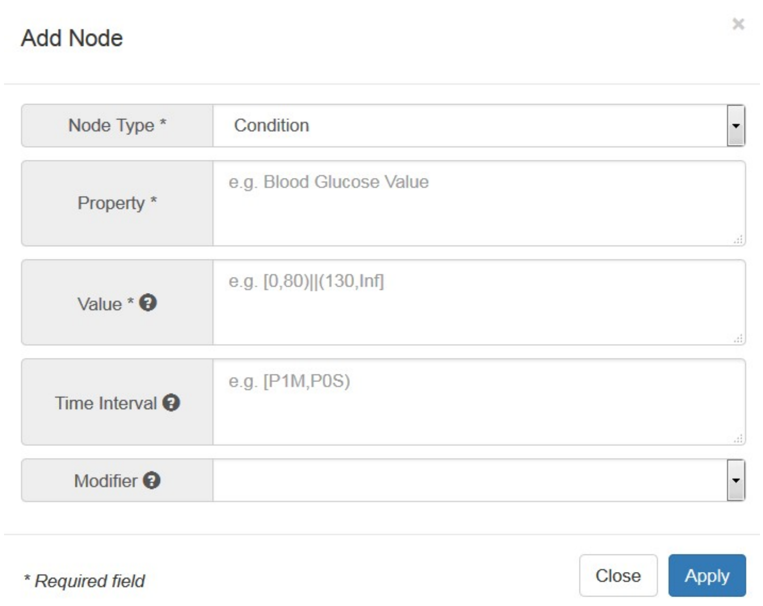
2. Nodes & Edges



@ALL: Some comment
Value Gap
Barrier Detection
Where do you have problems?
Insulin / Medication
What is the specific problem?
Dose too low
Specify the problem in detail
You are scared of having a hypoglycae...
Injecting 2 more units when not alone...
Recommendations
*
Low Dose Lesson

Figure 21 – Example Workflow

In order to add a new node click on "Add Node" and then click on an empty place in the drawing area where you want to initially place it. This opens a pop-up asking you to select one of the predefined node types (see Figure 22).
In order to edit an existing node click on the node, then click on "Edit Node". This opens the same pop-up with pre-filled data.



Add Node

Node Type *	Condition
Property *	e.g. Blood Glucose Value
Value *	e.g. [0,80)||(130,Inf]
Time Interval	e.g. [P1M,P0S)
Modifier	

** Required field*

Close Apply

Figure 22 – Add Node Pop-up

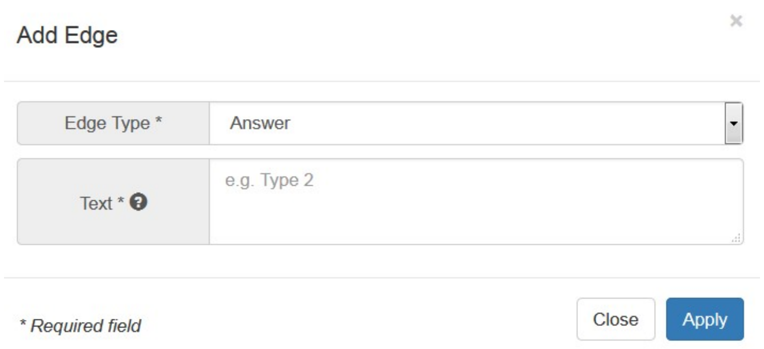
There are various node types with different purposes and input fields. In case the notations are not self-explanatory the help icons (question mark icons next to the field titles) can be used for further information. You can use <<ANSWER>> to reference to the answer of the previous node.
Important: All text input shall be provided in English only – internationalization will be handled separately.
The node types are as follows:

- Comment: Can be used to add comments when collaborating on a workflow (won't influence the actual workflow process)
  - Text (required): The comment text to be displayed
- Question: Can be used for any type of question
  - Text (required): The question text to be displayed to the user

    - Answer type (required): What kind of input is expected as an answer

- Content: Can be used to show recommendations, specific pages, static content, etc. to the user
    - Type (required): The content type (e.g. Recommendation, Goal, …)
    - Text or Link (required): The text to be presented or a link to the content. Multiple values can be separated using new lines
- Condition: Can be used to check specific conditions
    - Property (required): Which property to check (e.g. Blood Glucose Value). **Important: The property names are not fixed so please remain consistent within the same workflow**
    - Value (required): The value to evaluate against (textual or numeric). Multiple values can be separated using ||. Numeric intervals can be annotated using the mathematical bracket notation
    - Time Interval: By default the latest value will be checked. If you want to evaluate against a specific time range this range can be defined using the ISO 8601 notation
    - Modifier: By default the average value will be check when there are multiple numeric values. If you want to check the minimum/maximum this can be set here
- Trigger: Can be used to manually or periodically trigger a (sub-)workflow
    - Name (required): The name of a trigger can be used to manually trigger it using an “Action”
    - Time: Can be used to trigger it periodically (again using the ISO 8601 notation)
- Action: Can be used to manually activate specific triggers
    - Triggers (required): The name of the trigger to be activated

In order to add a new edge click on “Add Edge” and then draw the edge from one node to another by pressing the left mouse button on the source node, moving the cursor to the destination node and releasing the mouse button. This opens a pop-up asking you to select one of the predefined edge types (see Figure 23).
In order to edit an existing edge click on the edge, then click on “Edit Node”. You can now replace the endpoints of the edge. If you only want to edit the edge data move one of the endpoints slightly to the center of the current node. This opens the same pop-up with pre-filled data.


Add Edge

Edge Type *	Answer
Text *	e.g. Type 2

* Required field

Figure 23 – Add Edge Pop-up

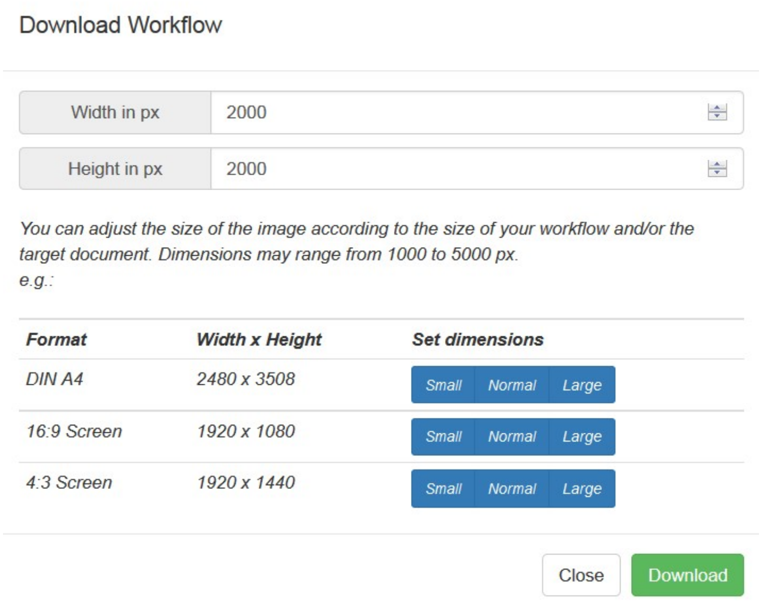
There are various edge types with different purposes and input fields. In case the notations are not self-explanatory the help icons (question mark icons next to the field titles) can be used for further information.

Important: All text input shall be provided in English only – internationalization will be handled separately.

The edge types are as follows:

- Answer: Can be used to forward to the next node if the response data of the preceding question matches
  - Text (required): The value to evaluate against (textual or numeric). An asterisk (*) can be used to annotate a “take this route if no other answer matches” edge. If you want to annotate multiple values use brackets and the logical operators || (OR) and && (AND). Numeric intervals can be annotated using the mathematical bracket notation
- Forward: Can be used to forward after the preceding node has finished

3. Download Workflow



Download Workflow

Width in px	2000
Height in px	2000

You can adjust the size of the image according to the size of your workflow and/or the target document. Dimensions may range from 1000 to 5000 px.
e.g.:

Format	*Width x Height*	*Set dimensions*
DIN A4	*2480 x 3508*	*Small Normal Large*
16:9 Screen	*1920 x 1080*	*Small Normal Large*
4:3 Screen	*1920 x 1440*	*Small Normal Large*

Figure 24 – Download Workflow Pop-up

Upon pressing the download icon (see item 6 in Figure 20) a pop-up is opened and prompts the user to input the desired dimensions of the final PNG image.
The dimensions may be entered manually or automatically by using the dimension table buttons.
Upon pressing the "Download" button the PNG image is created in a new tab (you may have to enable pop-ups when prompted to do so).
The resulting image can be saved via "Right click > Save image as…".