AMBIENT ASSISTED LIVING, AAL

JOINT PROGRAMME

ICT-BASED SOLUTIONS FOR ADVANCEMENT OF OLDER PERSONS' INDEPENDENCE AND PARTICIPATION IN THE "SELF-SERVE SOCIETY"

D3.3

Service Platform

Project acronym:	**GeTVivid**
Project full title:	**GeTVivid - Let's do things together**
Contract no.:	**AAL-2012-5-200**
Author:	**USG, ISOIN**
Dissemination:	**Public**

TABLE OF CONTENTS

TERMINOLOGY & ABBREVIATIONS **4**

1. EXECUTIVE SUMMARY **5**

1.1 LINK WITH THE OBJECTIVES OF THE PROJECT 5

1.2 STATE OF THE ART 5

2. SYSTEM CONCEPTION **7**

2.1 OUTLINE 7

2.2 META-ARCHITECTURE 7

2.3 ARCHITECTURE 11

2.3.1 Backend *11*

2.3.2 Frontend *12*

2.3.3 Client *12*

2.4 TECHNICAL COMPONENTS 12

2.4.1 Development environment *12*

2.4.2 Python dependencies *13*

2.4.3 Cascading Style Sheets *14*

2.4.4 Java Script *14*

2.4.5 MySQL *14*

3. WEB-UI **15**

3.1 STRUCTURING ELEMENTS 16

3.2 USER MANAGEMENT 17

3.3 ACCOUNT MANAGEMENT 18

3.4 SERVICE MANAGEMENT 19

3.5 APPOINTMENT MANAGEMENT 20

4. WEBSERVICES **23**

4.1 GENERAL OPERATION PRINCIPLE 23

4.2 SETUP 24

4.3 BASIC OAUTH FUNCTIONS 24

4.3.1 Request Token *25*

4.3.2 Refresh token *29*

4.4 ACS BACKEND ENDPOINTS 30

4.4.1 Tenant management *30*

4.4.2 User authentication *31*

4.4.3 Account management *31*

4.4.4 Service hierarchy *36*

4.4.5 Service management 36
4.4.6 Demand management 37
4.4.7 Appointment negotiation 38
4.4.8 Support functions 40
5. TASKS OVERVIEW 42
5.1 DEVELOPMENT OF MODULES FOR ACCESSING (EXTERNAL) INFORMATION SOURCES 42
5.2 BUSINESS PROCESS INTEGRATION 42
5.3 MANAGEMENT OF USER PROFILES, INTERFACES FOR TV AND MOBILE CLIENTS 42
5.4 BROADCAST PLAYOUT CENTRE COORDINATION AND INTERFACING 43
5.5 DEVELOPMENT AND DEPLOYMENT OF THE SERVICE PROVIDER FRONTEND 43
5.6 PROFILER MODULE 43
5.6.1 Profiler architecture 43
5.6.2 Business logic layer 45
5.6.3 Web services layer 47
5.6.4 Data model layer 49
6. OUTLOOK AND FURTHER DEVELOPMENT 50
REFERENCES 51

TERMINOLOGY & ABBREVIATIONS

ACS ..... Appointment Coordination System (synonym for Service Platform)

CSS..... Cascading Style Sheets

ERP ..... Enterprise Resource Planning

HTML..... Hypertext Markup Language

IWI..... Institute of Information Management

JS ..... Java Script

JSON ..... JavaScript Object Notation

resp. ..... Respectively

REST..... Representational State Transfer

SaaS..... Software as a Service

Sass..... Syntactically Awesome Style Sheets

SOA..... Service Oriented Architecture

UI..... User interface

1. EXECUTIVE SUMMARY

1.1 Link with the objectives of the project

As defined in the project's proposal, the service platform can be considered the central component of the emerging system, covering the backend for core functions and services. These functions and services comprise beside others the user account management, the main process logic for the mediation of informal and professional services, interfaces for customer-specific user-interfaces (UIs), and services for the business process integration with professional service providers.

The main task described by this document is task 3.4 (development of service platform). Requirements resp. functionalities are derived from task 2.2 (user requirements investigation), task 3.1 (system conception and architecture design), and task 5.2 (business requirements). In accordance with the proposal, the present document represents deliverable D3.3 in association with the prototype application.

In further deliberations, "Appointment Coordination System" (ACS) will be used as a synonym for service platform. The purpose of this document is to give insights regarding the orientation and structure of ACS, which is developed and maintained by the Institute of Information Management (IWI) at the University of St. Gallen. While trying to reveal as much information as possible, some aspects of the system are considered to be part of the actual intellectual property respectively unique selling proposition of ACS and the emerging solution and can hence not be covered in this document in detail due to the applied dissemination level.

1.2 State of the art

Looking at its core functionalities, ACS is a hosted multi-tenant marketplace solution for electronic service marketplaces. Based on and derived from the characteristics and functionalities of product related marketplaces, the central idea is to provide a platform that allows the handling of services just like every other product in the online environment. This idea, however, clearly has some limitations and is also the reason why an ordinary online-shop solution like Magento[1] is neither applicable nor (easily) adaptable. Listed below is a brief and not completed overview of differences that have to be considered when trying to bring services online:

- Delivery
  One of the most obvious and understandable differences between services and products is the availability and applicability of different delivery methods. Most products can easily be shipped and most parcel services have multiple escalation levels available that can be used if a recipient is not available. With services, on the other side, this is not always possible. There are a couple of services (e.g., home cleaning) that would at least theoretically allow the absence of the consumer, but for others (e.g., haircut) this is not possible. Hence, the basic "delivery method" for services has to focus on coordinating an appointment between consumer and provider. In the scientific environment this problem is called "uno-actu" principle and describes the situation, that production and consumption for services are often executed simultaneously.

[1] http://magento.com/

- Product respectively service specification
  In comparison to (most) products, services are harder to describe, as especially their price and the time needed in order to be fulfilled, are highly dependent on the particular case. The price for a haircut, to name just one example, can be affected my various variables, such as: gender (and therefore typical haircuts), hair length, included head massage, etc. The same applies to the description of some services.
- Payment
  Similar to and also connected with the specification of services is the question about possible payment strategies. Integrated payment is usually a convenient way for internet-based solutions; however, due to uncertainties regarding what is eventually delivered, it might be challenging to convince users of such functionalities. Additionally, to this rather individual perception is the "refundability" of services (most services cannot simply be returned as the effort has usually already been covered by the provider).

In order to deal with the mentioned and also further peculiarities of services and hence, requirements of service marketplaces, ACS offers a broad range of functionalities. Most of them are residing around the delivery respectively appointment coordination process with its variants, such as:

- Consumer and provider triggered request
  A normal request that is send from one party to the other (from the consumer to the provider or from the provider to the consumer of a service) and that might subsequently be considered as a non-committal request.
- Request without date (awaiting date suggestions from the other party)
  Technically a special case of the first one, during which the first request does not contain a specified date. This kind of request is useful if one party is significantly busier than the other (e.g., a dentist) and where the probability of picking a free time-slot (if not all appointments are maintained with ACS) is rather low.
- Single and multiple (alternative) date suggestions
  If a requested date is already blocked by one party or in case of incidents, an appointment can be re-coordinated by suggesting one or more alternative dates.
- Pre-negotiated requests
  In order to provide an additional value proposition especially for professional providers and also to streamline the appointment coordination process in general, ACS can be used to track and maintain appointments that have not been negotiated with the system itself. In this case, requests resp. appointments, which do not require any action from the other (consuming) party, can be added.
- Professional and informal services
  Professional services refer to "normal" providers that are providing paid services; informal services refer to private offers that are usual subject to other rewards.
- Demands
  As counterpart to service (and therefore offer) driven approaches, ACS allows the specification of a certain need (demand). This demand can subsequently be used for further appointment coordination processes, e.g., the suggestion of a maintained service (offer) in ACS or an ad-hoc answer just for the fulfilment of the demand.

During the following sections, details regarding the architectural design, technical components, and the web-service communication strategy are explained. For further details to the specific UIs please consult the respective deliverable.

2. SYSTEM CONCEPTION

2.1 Outline

This section describes the architecture of ACS. It describes, how a meta-architecture has been extended in order to fit to service paradigms and patterns. Based on the meta-architecture the actual implementation is developed.

2.2 Meta-Architecture

Defining or properly describing a mighty term like “architecture” is a bold venture. Industry hasn’t been able to identify or deduce a substantial definition and the same is true for the academic world (Gorton 2011, p. 2 ff.). However, when comparing some of the available definitions from standardization bodies (IEEE-SA Standards Board 2000) or textbooks (Bass et al. 2012; Garlan and Shaw 1993), it appears that they share a common core. The heart of this core states that “an architecture is the description of the set of components and the relationships between them” (Armour et al. 1999, p. 37). And while this definition can be applied to different fields (e.g., hardware, network, enterprise), each of them follows a set of distinct rules and models.

Against this background, we understand meta-architecture as an additional abstraction layer and thus as a “formal or semiformal Architecture Description Language […] that permits to describe particular systems, called architectures” (Smeda et al. 2005, p. 454). The purpose of a meta-architecture together with its advantages and disadvantages is widely discussed in literature and hence not covered here in detail. In summary, the main aspects are: a) standardizing the way an architecture is represented and therefore making it comparable, b) being guided by approved principles and therefore improving quality, and c) facilitating a clear structure with deliberated dependencies and therefore allowing for further development and adaptions.

Associated with the nature of the project, the development is expected to be highly agile and driven by adaptions and changes (e.g., in order to deal with third-party systems or as a response to end-user tests). The overall architecture as described in Deliverable 3.1 is distributed and service-oriented. Evidently, the benefit of certain meta-architectures is limited to specific application areas respectively to particular characteristics. To account for these situational conditions, STCBMER (Maciaszek et al. 2014) meta-architecture is applied. STCBMER is based on PCBMER, which again borrowed “the names of the external tiers […] from the Core J2EE Framework” (Maciaszek 2006, p. 9). The abbreviation PCBMER stands for the 6 layers of the meta-architecture that is shown in Figure 1. A short description of the respective competence of the layer can be found thereafter.

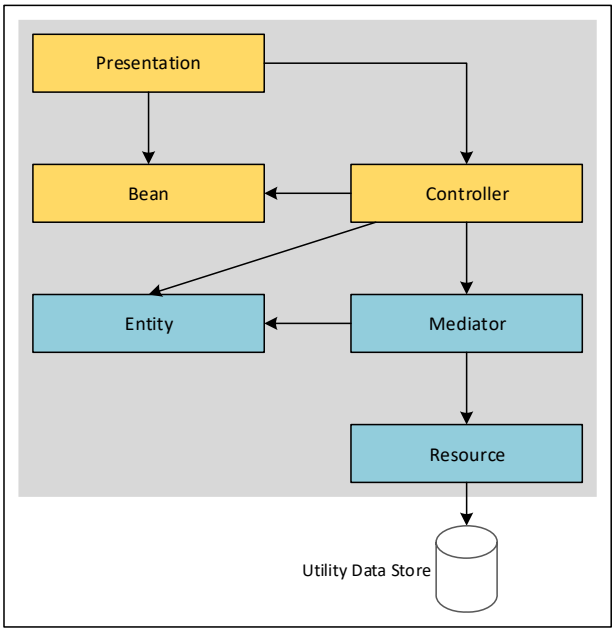


Figure 1: PCBMER meta-architecture

Description of the PCBMER layers (Maciaszek 2006):

Presentation: The Presentation layer is responsible for all questions directly related to the user-interaction and hence, the UI. Since the scope is limited to the representation towards a user, data-management tasks reside not within the Presentation layer even if data-management functions (e.g., an update) might require actions here.

Bean: The Bean layer encompasses all data objects or classes that are meant to be handled by the Presentation layer. Arising thereby are two sources for Bean data, entity objects and data entered by the user. In contrast to the Entity layer, information handled by a Bean might be implicit and therefore not represented (directly) in an Entity.

Controller: The Controller represents the main part of the application logic. It is responsible for the translation of Entities to Beans upon requests from the Presentation layer.

Entity: The Entity layer is used by the Controller and Mediator. Entities can be understood as a translation service between the front-end resp. front-end related layers and actual persistence services.

Mediator: The Mediator layer – according to its name – mediates between Entity and Resource classes. It covers business rules, business objects and caching. It can be understood as isolation layer between the Entity and Resource elements.

Resource: The Resource layer is the touchpoint for communication with external persistence units (e.g., database or web service).

The diffusion of service oriented architectures (SOA) and software as a service (SaaS) has raised new challenges to the field of architectural design, having the increased complexity due to heterogeneous integrations, more volatile requirements, and the urge for faster progression as only three of them. Following the basic idea of SaaS, using such services can help to focus on core competencies or problems and – at least in some cases – to gather

freedom due to the exchangeability. For the architecture of the service itself, this implies additional challenges in terms of adaptability. In order to cope with these changes, the meta-architecture STCBMER (Maciaszek et al. 2014) represents an adapted version of PCBMER. It consists of seven layers that are grouped into three main modules (Figure 2), whereas the three modules can be physically distributed. Most of the extensions have been developed against the background of ACS, which is why there might be application-specific design decisions.

Resource: The Resource layer represents the interface for accessing the actual persistence unit. Depending on the underlying technology (e.g., MySQL vs. web service), it contains tools for database communication, session management, or transactions.

Entity: The Entity layer consists usually of a set of business classes that are mapped from one or many elements of the Resource layer (e.g., database tables). Entity classes are, however, more than just mapped database tables; they are inherently connected with the business logic. This separation can also be seen in the sub-layers, Entity Object and Entity Object Adapter. The Entity Object holds the business entities that are mapped from data sources (e.g., via Object-Relational Mapping). Entity Object Adapters are classes that represent serialized Entity Objects that are ready to be sent via a web service. Technically, an automated translation resp. serialization would be possible. However, Entity Object Adapters are able to filter attributes and hence are able to decide about the visibility of attributes to external applications.

Mediator: The Mediator layer is responsible for business rules, business objects, and caching and offers CRUD functionalities on Entity Objects. Technically, it can be implemented as directly accessible methods or a set of web-service functions. The available functionalities that scope the underlying business rules, however, should be defined in the Business View sub-package. The routing to the respective function is declared in the Business Web Service Definition package. In collaboration with the Entity Object Adapter, the Entity Serializer covers the transport-medium specific serialization.

Controller: The Controller is the Mediator's counter-part, covering the applications logic defined by a set of functions or web-services. Each function or web-service is called a view (Application View) and covers the mapping between Entity Object Adapter and Bean Objects. Views use the Application Web Service Connector for communication and orchestration; definitions are provided by the Application Web Service Definition sub-package.

Bean: The Bean layer contains a set of application object classes that are meant to be handled in the context of front-end preparation. Depending on their purpose they can be mapped and serialized by the Bean Serializer resp. the Bean Object Adapter.

Template: The Template layer is responsible for the translation of Bean Objects resp., e.g., the serialized version of Bean Objects into a presentable format. The possible formats are not limited and range from web-based UIs to ordinary ones. There is also no further constraint regarding the level of generics that is applied here, which means that both, "hard-coded" UIs as well as approaches like Mako templates can be used here.

Smart-Client: The Smart-Client layer represents a front-end specific implementation for the user interaction. It follows the Model View ViewModel (MVVM) pattern.

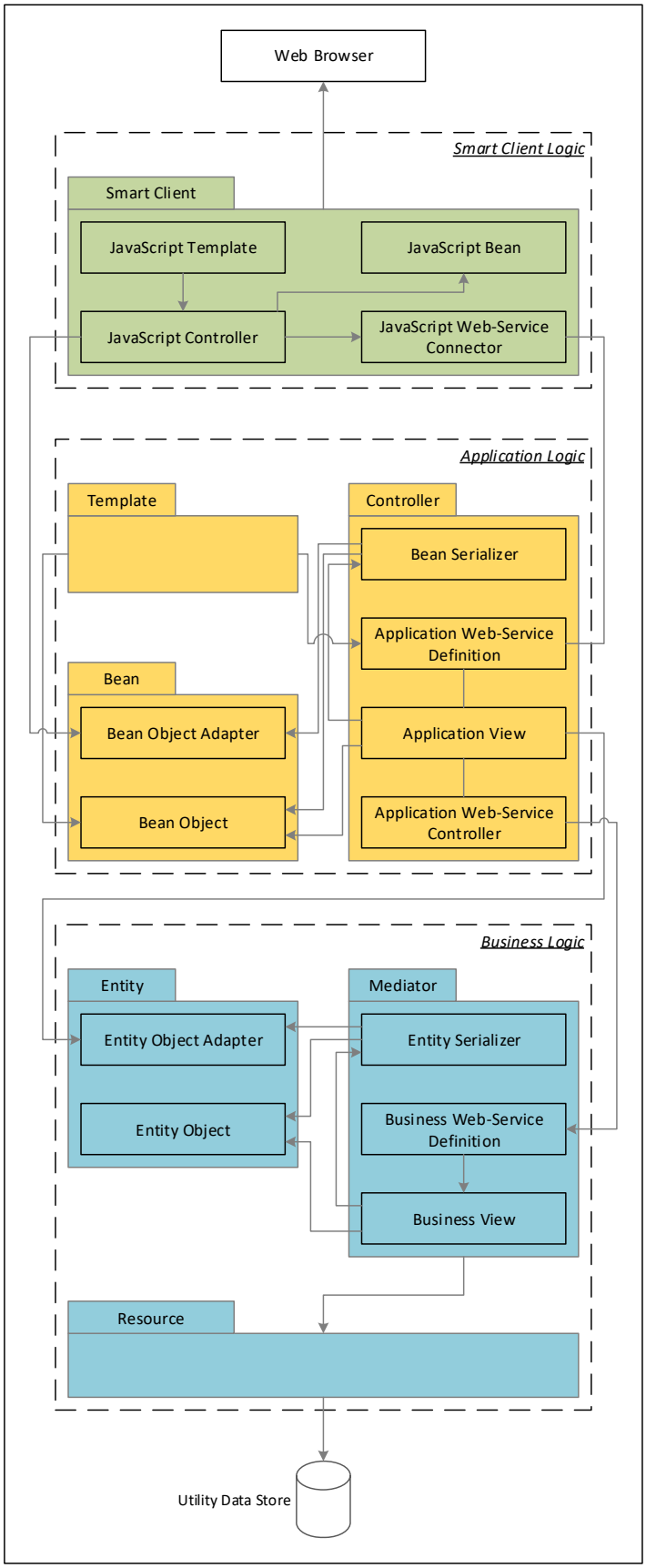


Figure 2: STCBMER meta-architecture

2.3 Architecture

The actual architecture resp. implementation of ACS follows the guidelines defined by STCBMER. The structure – as shown in Figure 3 – can be divided into three or five parts (depending on the applied level of detail).

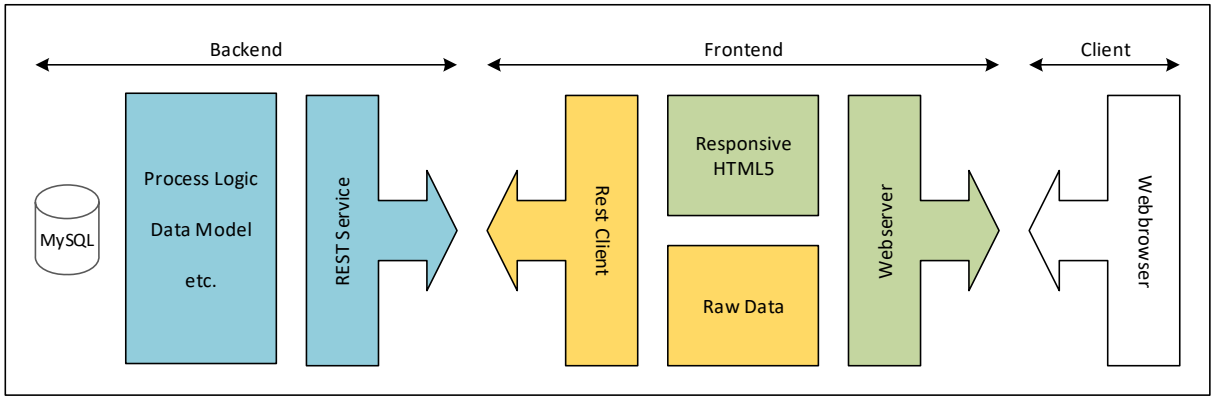


Figure 3: ACS architecture

On a higher level, there are three components resp. layers involved when it comes to using ACS: a decoupled backend, at least one frontend, and – depending on the used technology – the client. The backend represents the actual core of the system, encapsulating data storage, process logic, and web-services and is designed to be a single instance for a certain user network. The frontend-layer, on the other side, is to be understood as a collection of multiple user interfaces connected to a single backend instance, such as the normal / standard ACS web-interface or – with respect to GeTVivid – the HbbTV and mobile user interface. The client-layer as last tier is not necessarily a part of the system and is designed – if needed – in order to visualize what has been calculated on a frontend-layer instance. For a simple web-interface this might be a standard browser. However, there are other scenarios imaginable where this layer contains, e.g., container-apps or embedded functionalities.

2.3.1 Backend

In order to follow the principles of modularity, adaptability, and extensibility, the main components resp. layers of ACS are loosely coupled. The first and most important layer is the backend, which contains among others the main process logic of the application. Parts of this process logic might also be available in frontends in a redundant manner, but the power of ultimate decision about the validity of process steps is resided in the backend. Whenever information has to be stored resp. made persisted or has to be accessed, a data object model approach is used in order to access MySQL resources. In order to seal both, data storage and process logic, off, the only way to access backend resources is via REST web-services. This interface, however, is not completely open to all possible applications (and hence “frontends”) at the moment, but limited to pre-registered ones. Additional information might be found in section 3.

2.3.2 Frontend

ACS supports multiple frontends for a single tenant, whereas a tenant represents, e.g., a project like GeTVivid. One frontend that is available by default is the standard ACS web-frontend. It includes workspaces for both, (professional) providers and consumers. However, in the context of GeTVivid the web-frontend will solely be used by professionals. Generally, a frontend can be considered an UI-building control, which may utilize all available REST web-service functionalities and translate the results resp. the data into usable representations. In terms of the standard web-frontend this means, that from REST requests resulting data is transformed into responsive HTML5 code which is provided to the consumer via a webserver. In parallel to this approach, raw datasets are provided for asynchronous requests that are rendered within the client application. The design and development of other GeTVivid specific frontends (HbbTV and mobile device) are described in the respective Deliverable (D3.2 and D4.1).

2.3.3 Client

The third layer of ACS, the client application, should be understood in a more abstract manner. In contrast to other components, this layer might in some cases just be a standard application, e.g., a web browser. However, in the context of GeTVivid it is extended by an application (possibly even an embedded one) running on HbbTV or mobile devices, enabling channel and program independent functionalities.

2.4 Technical components

2.4.1 Development environment

In order to develop a system that is as homogenous as possible regarding its structure while following and fulfilling the technical description of work packages, computer-aided software-engineering is applied. Therefore “Visual Paradigm for UML 11.0” is used for modelling and architecture related tasks. Where possible and useful (in terms of time saving and applicability), stubs are created automatically. The actual programming is conducted in Python with “PyCharm”.

For a continuous integration strategy a git repository with Jenkins is used. Development tasks are implemented during Sprints on local DEV environments and tested by Unit-Tests. Subsequently a Test instance is used for frontend tests after which the development results are deployed to a Demo instance. The Demo instance is primarily used for the integration with other sub-systems of the project. After a successful integration, country specific field-trials instance are loaded with the final results (see Figure 4 for the whole process). The implementation respectively realization of the web-frontend is described with Sass (Syntactically Awesome Style Sheets) and compiled to CSS during the testing and deployment process.

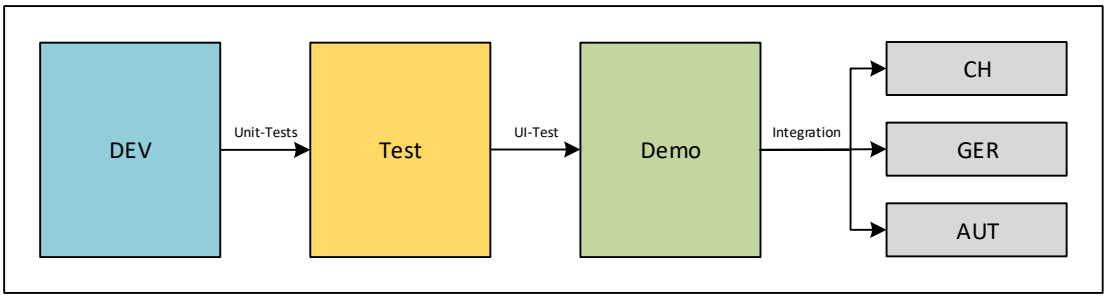


Figure 4: Integration strategy

2.4.2 Python dependencies

The following list shows a brief overview of the most important python libraries that are used or applied in the development process resp. that have been integrated into ACS.

[python2.7]	Python version 2.7
[pyramid]	Framework for developing web applications with Python
[SQLAlchemy]	Object-relational mapping framework for Python for avoiding the object-relational impedance mismatch[2]
[transaction]	Extension for securing database requests based on transactions (roll-back functionality)
[waitress]	Python based web server gateway interface
[mysql-python]	MySQL interface for Python
[nose]	Unit test discovery and execution framework
[coverage]	Extension for the testing environment allowing coverage tests
[sphinx]	Integrated documentation tool
[cornice]	REST framework for Python
[pyramid_mailer]	Mail-extension for the pyramid framework
[Babel]	Multi-language respectively internationalization library
[pycrypto]	Python cryptography framework
[valedictory]	Data validation library, used especially for user input
[whoosh]	Full-text indexing and searching library
[icalendar]	iCal support
[Pillow]	Python image processing library (e.g., in order to calculate thumbnails)
[pytz]	Date time and time zone conversions
[Mako]	Template engine (default language of Pyramid)
[dogpile]	Search engine fetching google results

[2] Problems referring to saving objects of object-oriented programming languages in relational databases

2.4.3 Cascading Style Sheets

ACS's standard web-based frontend is a responsive HTML5 component. The selection of this set of technologies is based on two reasons: First, HTML is the overall standard for web- respectively browser-related applications and version 5 the latest release. Second, responsiveness has been used in order to address certain peculiarities of different devices (standard computer, smartphones, etc.) without eventually developing multiple UIs. The responsiveness of the UI is based on Cascading Style Sheets (CSS), or to be more precisely, on boostrap 3.1. Bootstrap is a free collection of different tools, simplifying web-development with HTML and CSS templates (that have already been adapted to various browsers). Additionally, Sass is used for the declaration of UI elements in nested rules.

2.4.4 Java Script

JavaScript (JS) is used in order to increase the responsiveness of the web-based frontend even further and facilitate not only higher performance (due to less full page refreshes) but also a user experience closer to rich-clients. At the beginning of the ACS development, jquery has been used as JS library. However, due to problems with components in terms of responsiveness and availability for mobile-devices, jquery will be replaced by angular.js and Bootstrap related JS components.

2.4.5 MySQL

MySQL 5.6 is used as database system. All functionalities that are not covered by the standard MySQL are realized via additional libraries (e.g., transaction as introduced in section 2.4.2).

3. WEB-UI

Following the description and specification in section 2.3 respectively 2.3.2, Web-UI refers to ACS's built-in web-based user-interface. The Web-UI consists of a CSS propped responsive HTML website that can be used on multiple device sizes (e.g., desktop computer, notebook, tablet, smartphone, etc.). For mobile devices running on Android and iOS, native container apps are available, as well. Figure 5 gives an overview of the most important available functionalities, which are shown in screenshots thereafter.

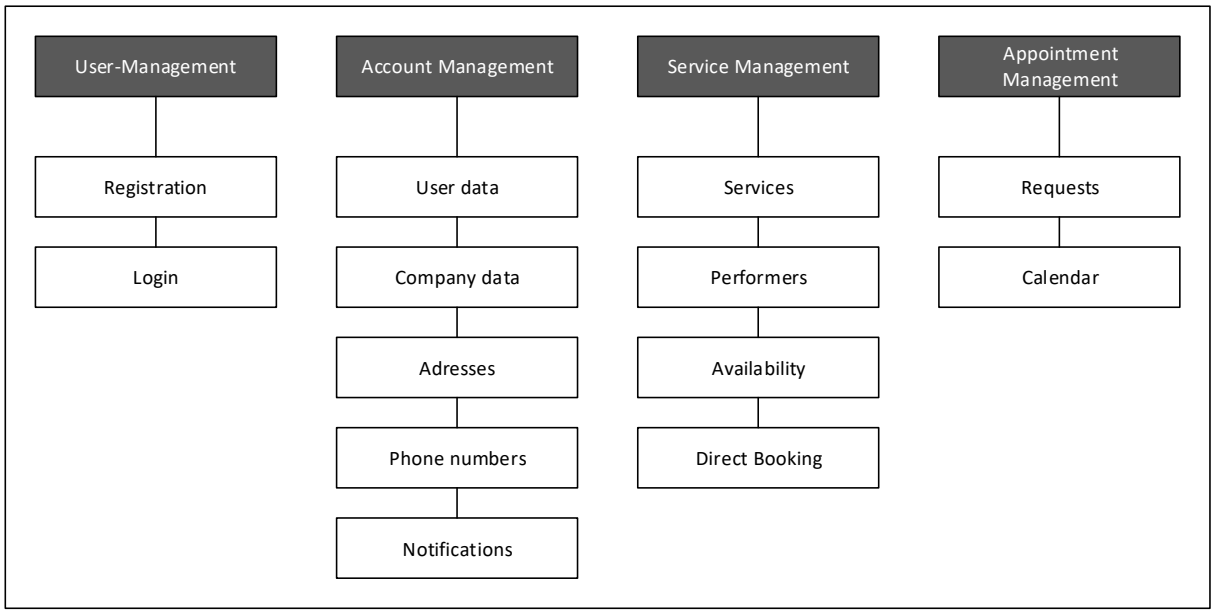


Figure 5: Web-UI functionalities

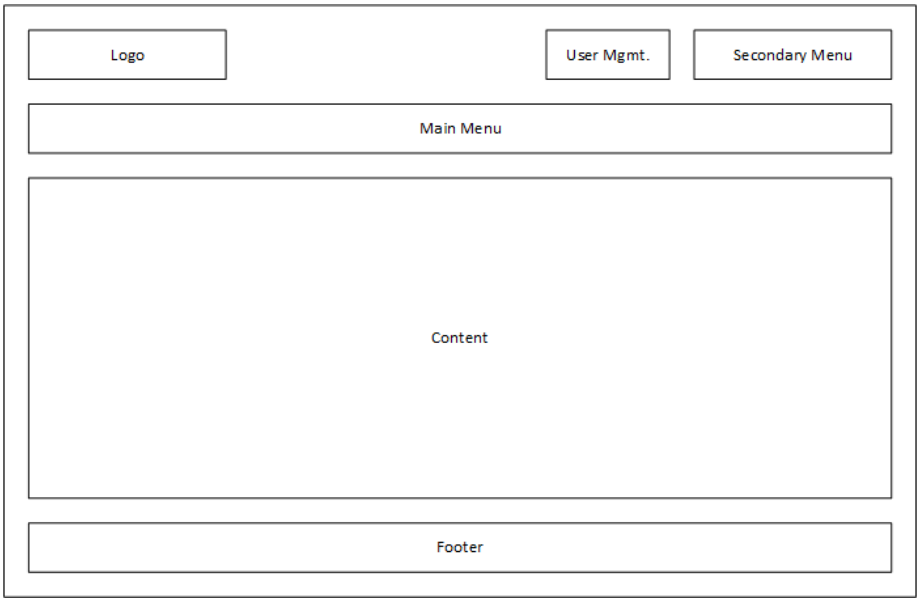


Figure 6: ACS Web-UI Structure

The basic UI structure is shown in Figure 6. In front of a page filling background image, colour-coded elements frame the four main levels of the UI. The first level contains a logo, the main aspects of the user management and the secondary menu. The second level consists primarily of the first-level respectively the main menu. Beneath the mentioned levels, the main content area is followed by a footer whereas both of them have an adjustable height. All used texts and captions are integrated via gettext (internationalization and localization system), thereby preparing the system for a multi-language context. The following screenshots, however, are kept in German since this is the language used in all field-trials and therefor best tested.

3.1 Structuring Elements

Figure 7 shows an overview of the main structuring elements, as already presented in a more abstract way in Figure 6. The colour has been adapted to GeTVivid's CI whilst the background image is a placeholder. The main section, market by a less transparent background, is adapted in height due to the integrated content. The same is true for the footer, which contains a link list with basic information for the providers.

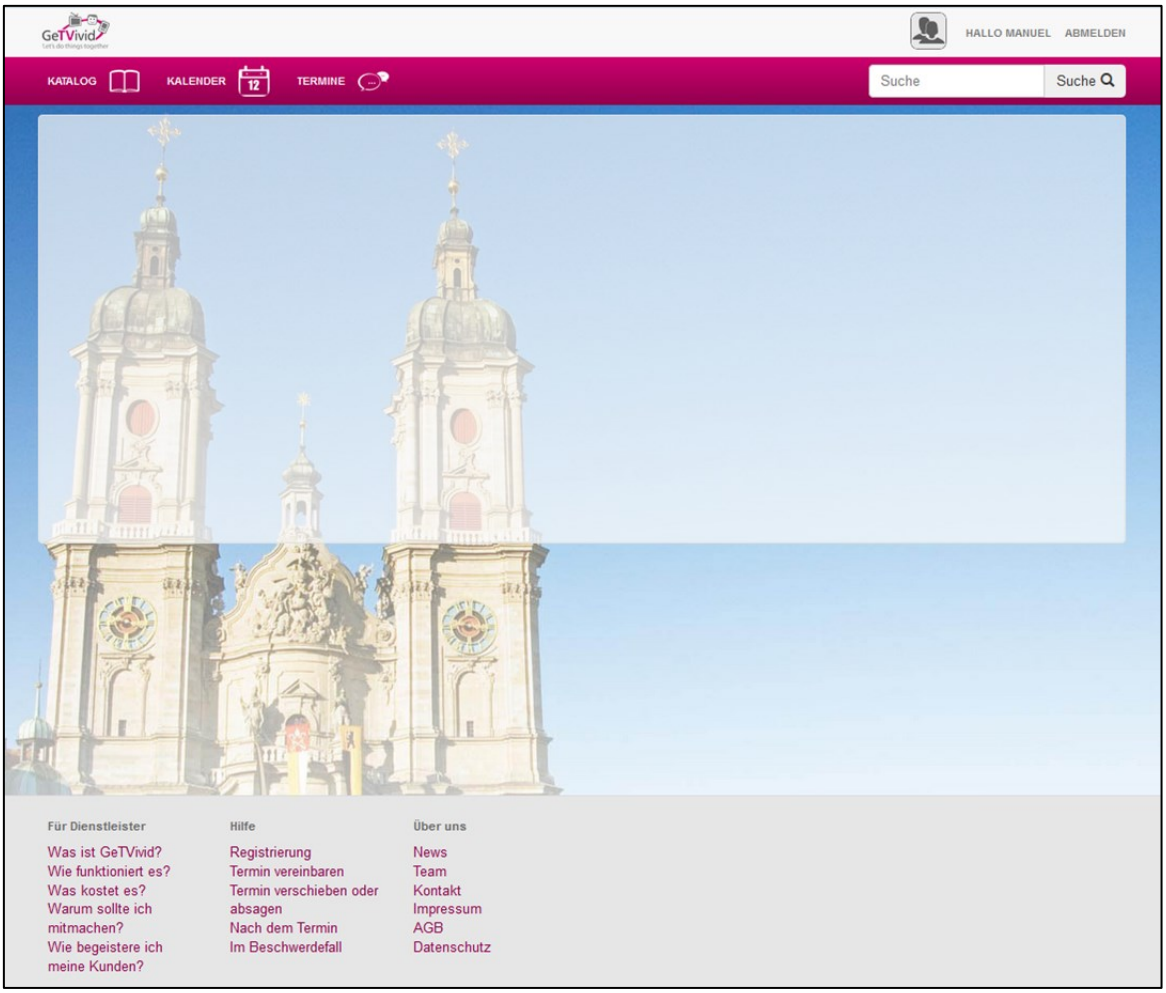


Figure 7: ACS Web-UI Structuring Elements

3.2 User Management

The user management part consists of two main functions: registration and login. Even though basically available for both groups, customers and providers, the description focuses solely on the provider's side. The registration process can be started from the secondary menu. As first interaction, the user has to choose the correct role (Figure 8) before the actual registration form is shown. The form itself shows input fields for the most important details that have to be submitted (Figure 9).

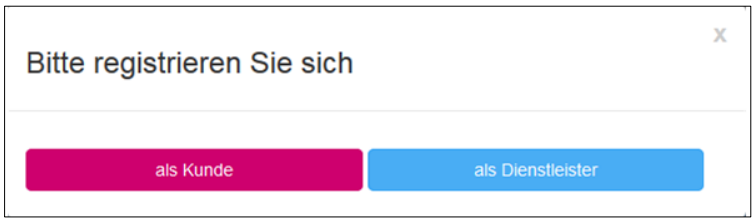


Figure 8: Registration Role Selection

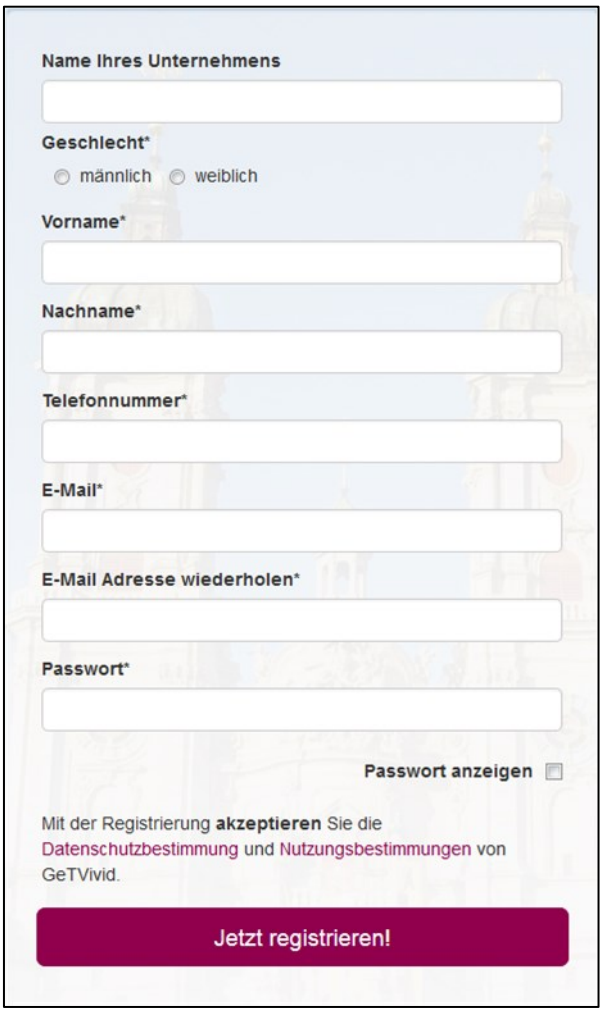


Figure 9: Registration Form

3.3 Account Management

The user data management view (Figure 10) gives an overview of the most important account-related datasets. Available information is grouped into personal data, phone numbers, SMS notifications and addresses. Details can be maintained (e.g., added, updated, deleted) in the respective sub-menu.

Besides standard e-mail notifications, SMS based notifications are supported as well. In order to avoid an annoying flood of incoming short-messages, the level of demanded notifications can be controlled by the user. Address administration is supplemented with a Google maps extract for additional feedback.

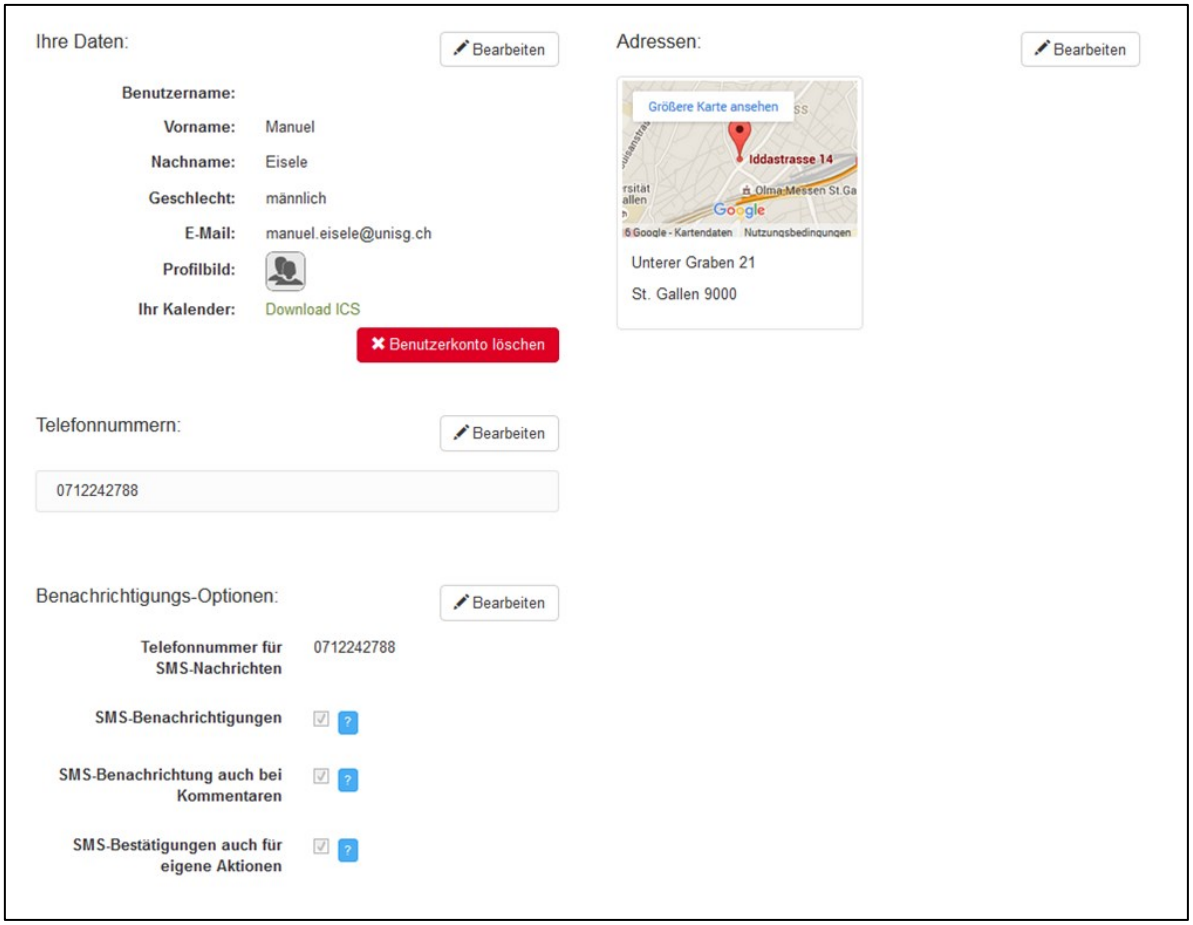


Figure 10: User Data Management

Company related information is prepared in a similar manner as shown in Figure 11. Analogous to the user data management view, company details are grouped into general details, phone numbers and addresses. Additional company related datasets like services and employees are further described in section 3.4.

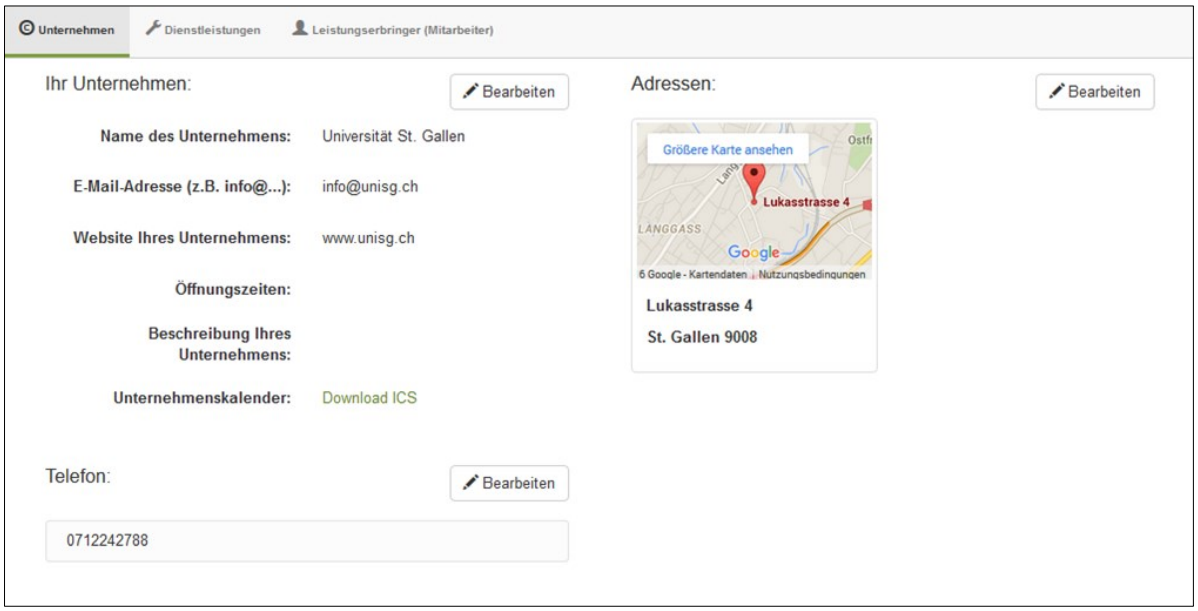


Figure 11: Company Data Management

3.4 Service Management

The service management section allows CRUD functionalities for (professional) services offered via ACS. Services are defined and edited in a simple form with all required details, like: name, description, pricing, catalogue category, performers able to fulfil it, etc. (Figure 12).

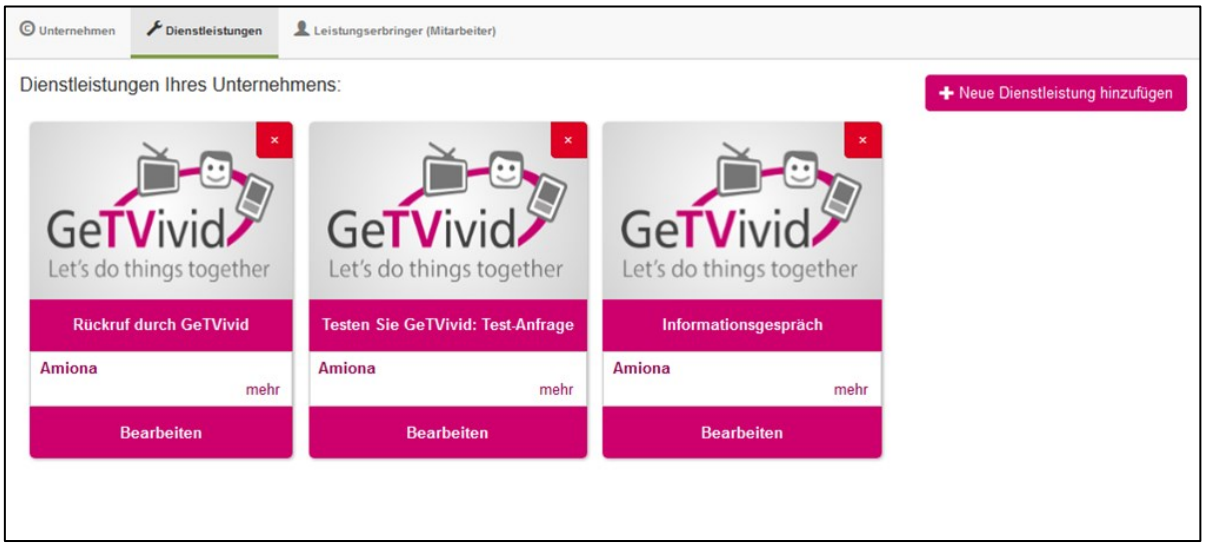


Figure 12: Service Management

In order to support the provider's workflows as far as possible, employees / performers can be maintained in ACS as well (Figure 13). This allows for three main scenarios:

1) As a provider / manager, assigning a request to a performer / employee.
2) Having a request automatically assigned to a certain performer, if he / she is the only person marked as performer for the requested service.
3) Automatically assigning requests to performers based on their respective availability (e.g., working hours).

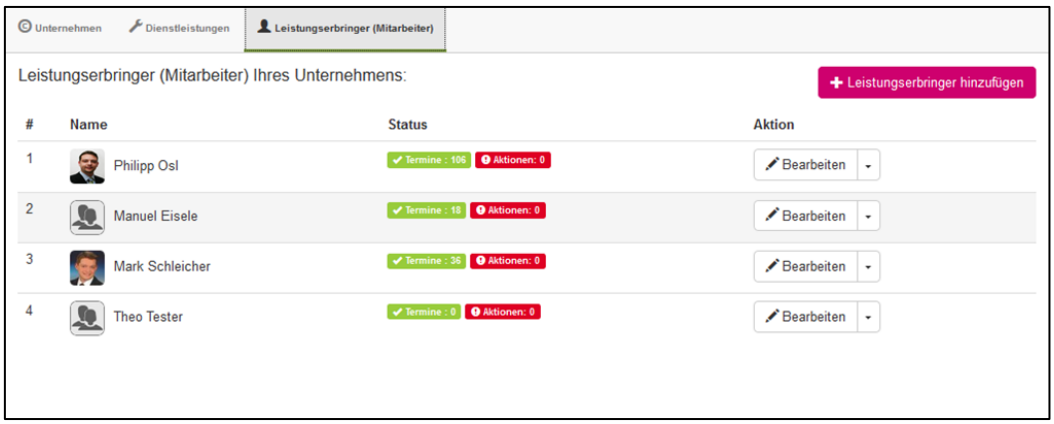


Figure 13: Employee / Performer Management

3.5 Appointment Management

In the appointments section, a list of all received requests and negotiated appointments is shown (Figure 14). By default, the list is ordered by currentness; however, this ordering is overruled by requests requiring some action from the user (e.g., an answer in terms of accepting, declining, or rescheduling an appointment request). Additionally, user-specific filters can be applied (e.g., all appointments from today).

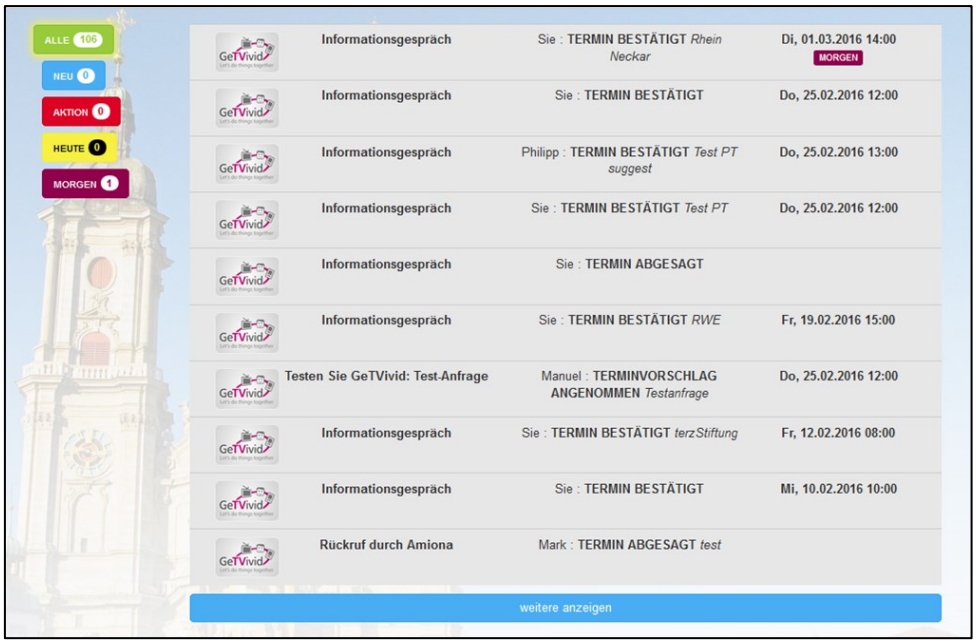


Figure 14: Appointment Overview

In the details view of an appointment request, all information related to a request plus the available actions are displayed (Figure 15). Providers can use this presentation in order to gain an overview of the request (e.g., client, related address, etc.) and answer the request.

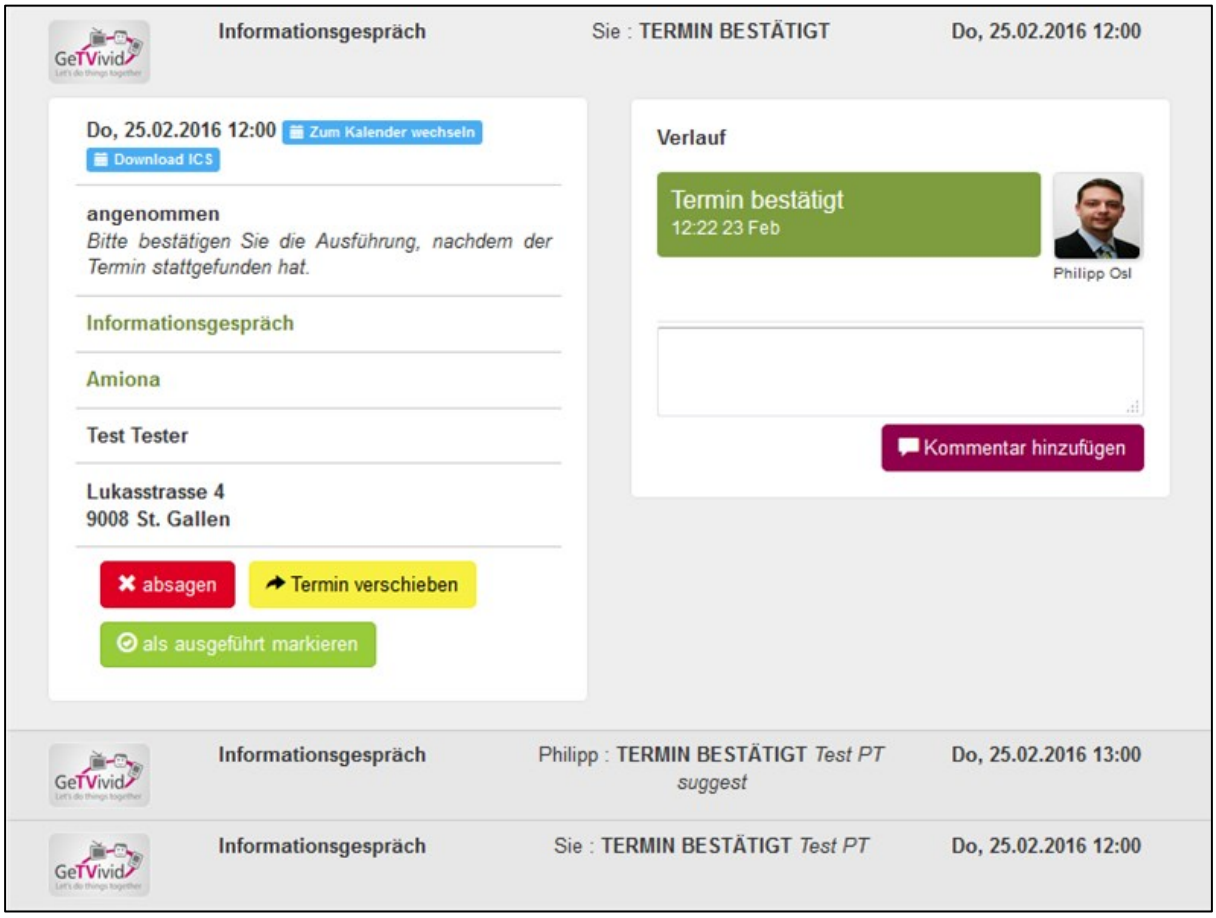


Figure 15: Appointment Details

The calendar view (Figure 16) shows basically the same information as the appointment overview list, but brought to a hard chronological order. Clicking on a calendar entry refers to a details view similar to the one above.

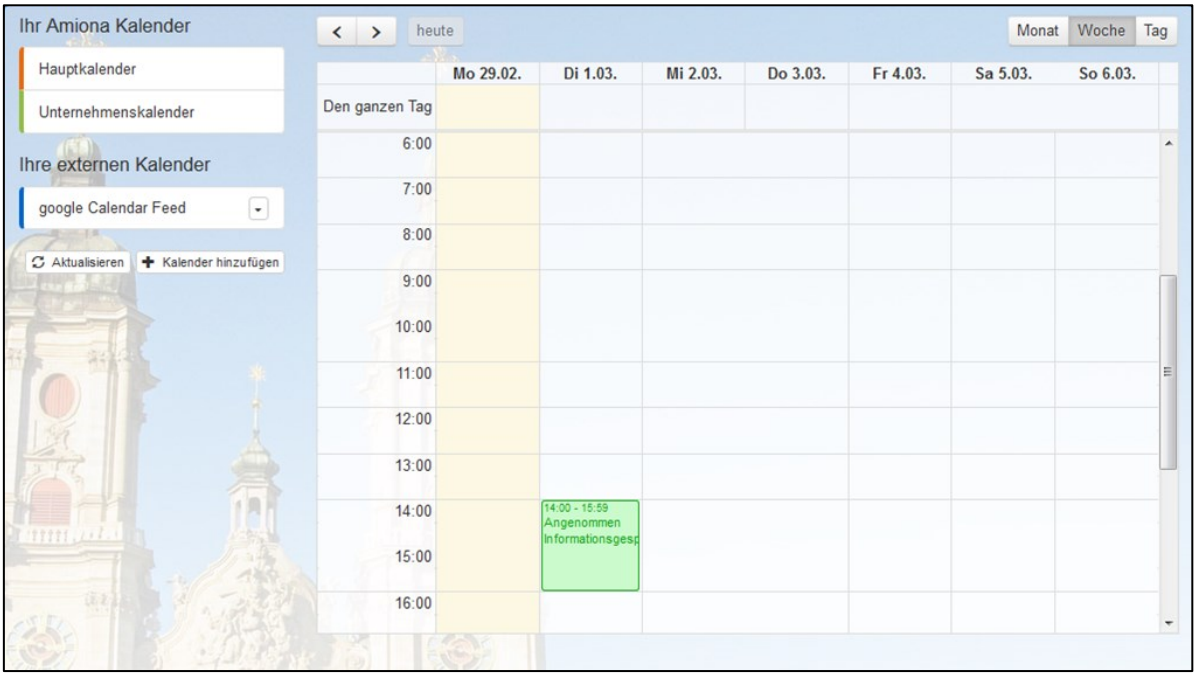


Figure 16: Calendar View

4. WEBSERVICES

4.1 General operation principle

All web-services area realised as representational state transfer (REST) services with JavaScript Object Notation (JSON) as representation language. Safety related and privacy relevant endpoints are secured via OAuth 2.0. The ACS REST interface can be used in order to implement a secure delegated access to the resources resp. the data of a certain ACS user. Since all requests have to be made on behalf of such a user, the user's credentials are required. The inversion of this argument is that the OAuth REST services cannot be used in order to perform standard platform operations like requesting the name for a given user ID (except it is the ID of the user that is represented by the OAuth access token).

The basic procedure is displayed in Figure 17. In this figure, "User" represents the actual ACS user, "Application" any 3rd party application that is about to use the OAuth interface in order to request data resp. execute functions on behalf of the user and "ACS" the standard ACS backend.

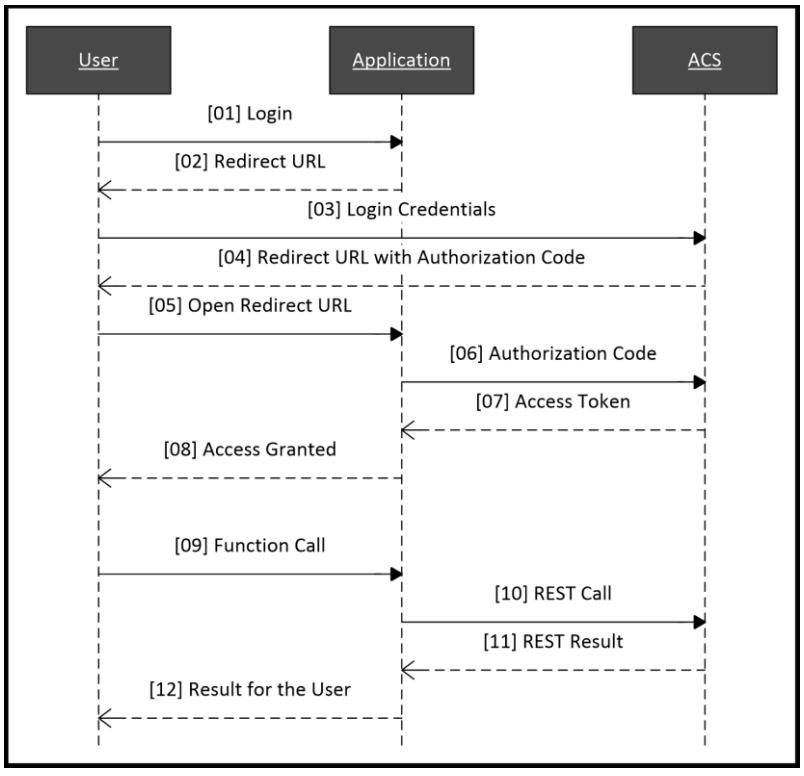


Figure 17: Basic OAuth Procedure

The request and response sequence is defined as follows:

[01] The login action is somehow requested by the user, e.g., by clicking on a login button in the 3rd party application.

[02] The User is redirected by the 3rd party application to the login form of ACS. The redirect URL contains i. a. the client resp. application ID and the corresponding secret, which identifies the 3rd party application.

[03] Login credentials (username and password) are directly sent from the user to ACS.

[04] In case of a successful authentication, a redirect URL (in most cases referencing the 3rd party application again) is sent to the user. The redirect URL contains a one-time authorization code.

[05] By opening the redirect URL, the authorization code is delivered to the 3rd party application as GET parameter.

[06] The 3rd party application is to extract the authorization code. A temporary access token can now be requested by the 3rd party application in exchange for the one-time authorization code.

[07] The access token is provided to the 3rd party application.

[08] Authentication process resp. request token process is finished. If the 3rd party application is only interested in some kind of user-database for login purposes, the login is actually finished at this point.

[09] At any later point of time, a user might use a function of the 3rd party application that requires some ACS-data or -help.

[10] The 3rd party application can now use the REST services in connection with the access token.

[11] The REST result is delivered to the 3rd party application.

[12] The final result is delivered from the 3rd party application to the user.

4.2 Setup

Before the OAuth services of ACS can be used, the 3rd party application that is supposed to communicate to the REST services has to be registered in order to receive access credentials as shown in Table 1:

APPLICATION DETAILS			
Client/Application ID	__client_id__	**Client/App. Secret**	__client_secret__
Redirect URI	__redirect_uri__		
Scope	DEFAULT	**Response type**	Code
API	__api_url__		
Domain	Test	**Domain ID**	1
TEST USER DETAILS			
Consumer 1	User: user_1	Password: pw_1	
Consumer 2	User: user_2	Password: pw_2	

Table 1: OAuth Application Details

4.3 Basic OAuth functions

For the detailed description of OAuth functionalities, the message resp. communication flow of Figure 17 is used as reference. Numbers in squared brackets refer to the respective message as described above.

4.3.1 Request Token

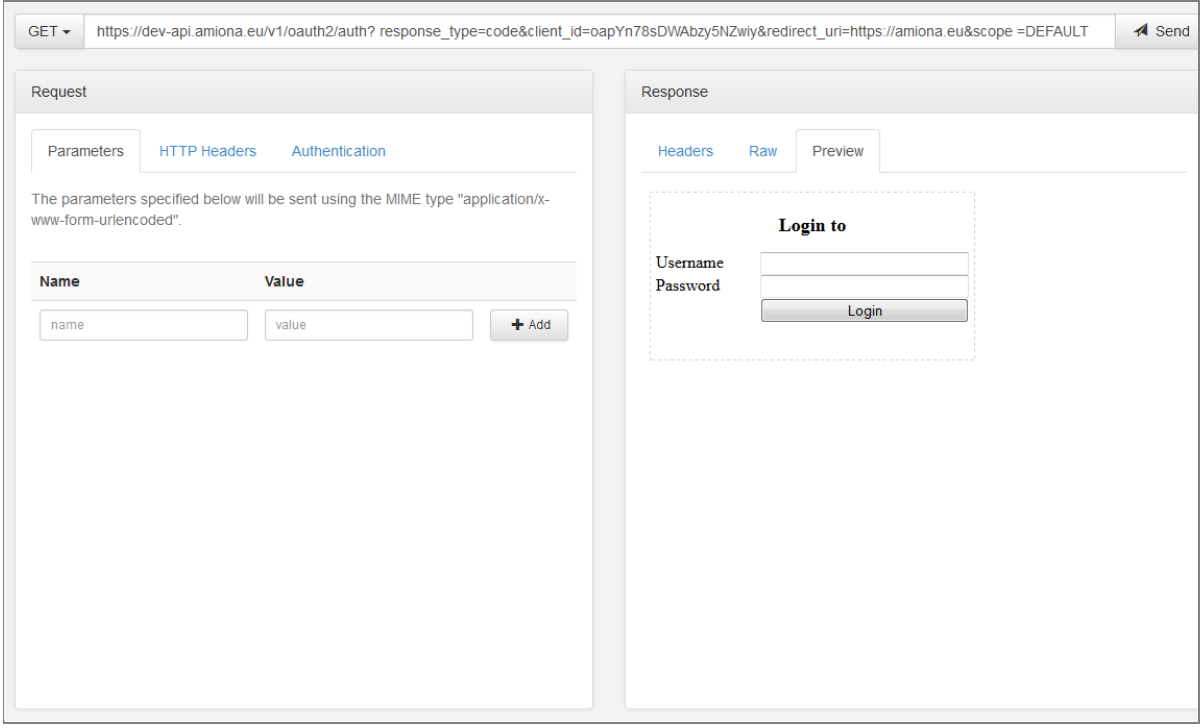
Requesting an access token is the first step of the OAuth 2.0 authorization process (Figure 17). According to the changes of version 2.0, an initial get token request from the 3rd party application to the ACS OAuth server is neither required nor possible.

In order to request an OAuth access token, the user has to be redirected to the ACS login form [02], as the exchange of user credentials is based in a direct connection between the user and ACS [03]. For this call – and also for all upcoming ones – the API URL is used as basis and enhanced by the directory. If the user enters his/her credentials correctly, a redirect URL is sent back to the user [04]. By opening the URL, a one-time authorization token is delivered to the 3rd party application [05]. The authorization token together with the 3rd party application's credentials is submitted to the OAuth server [06] and thereby exchanged into a durable access token for a certain user [07].

4.3.1.1 Message [02]: Redirect user to the ACS login form

REDIRECT URL			
In order to start the authentication process, the user has to be redirected to the ACS login form. This redirect URL has to contain all parameters that are listed below. After a successful validation of the credentials, the user is redirected to the redirect_uri. The redirect URI is currently defined in the ACS database. However, in order to achieve a higher level of security, the initial (user's) redirect to the login-form [02] has to contain the same redirect URI as GET parameter.			
Directory	`/v1/oauth2/auth`	**Method**	`GET ¦ PARAM`
PARAMETERS			
`response_type`	Type of response. At the moment, only "code" is available.		
`client_id`	ID of the client application (not of the user).		
`redirect_uri`	URI the user is redirected to, in case of a successful login. The URI is pre-set as database entry. However, for additional security this pre-set value has to be confirmed by submitting the same URI as GET parameter.		
`scope`	Scope of functionalities that should be covered by the token; at the moment only "DEFAULT" is available.		

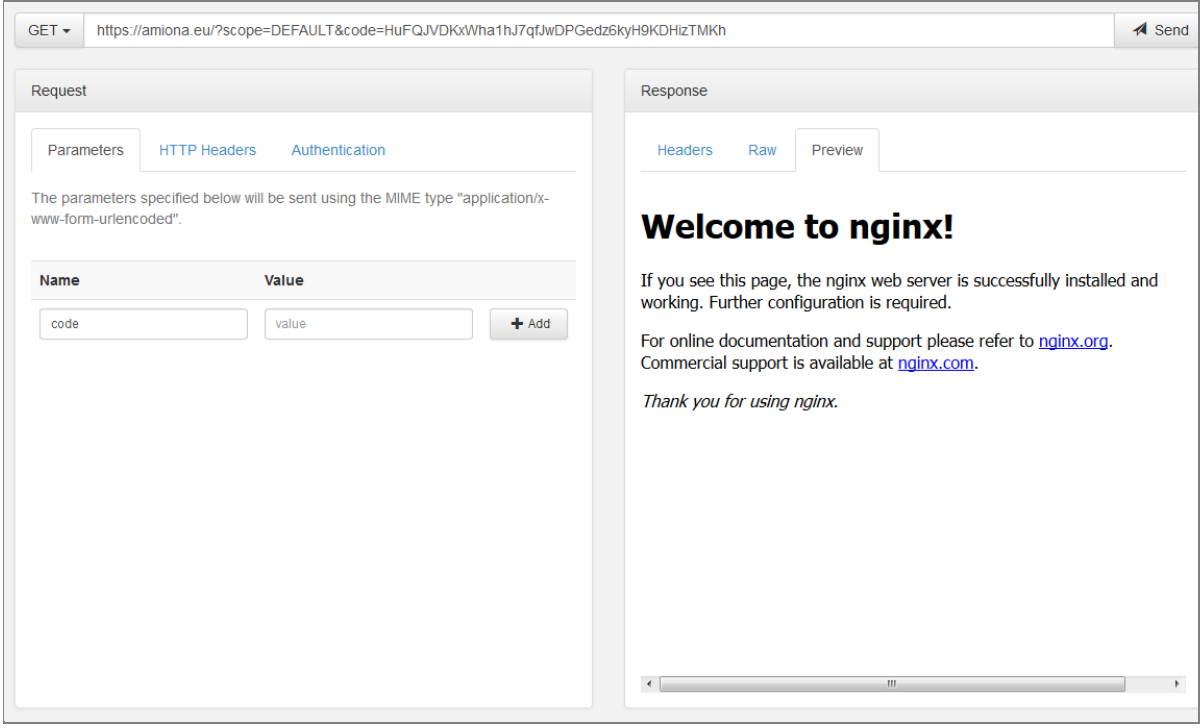
SAMPLE



4.3.1.2 Message [04]: Redirect authenticated user back to redirect_uri

RESPONSE			
In case of a successful login, the user is redirected to redirect_uri. The redirect_uri will contain the “scope” and an authentication “code” as GET parameters.			
URI	`redirect_uri + parameters`	**Method**	`GET`

PARAMETERS	
`scope`	The scope for the access that was granted by the user.
`code`	One-time authentication code, which can be used to be exchanged for a durable access code.

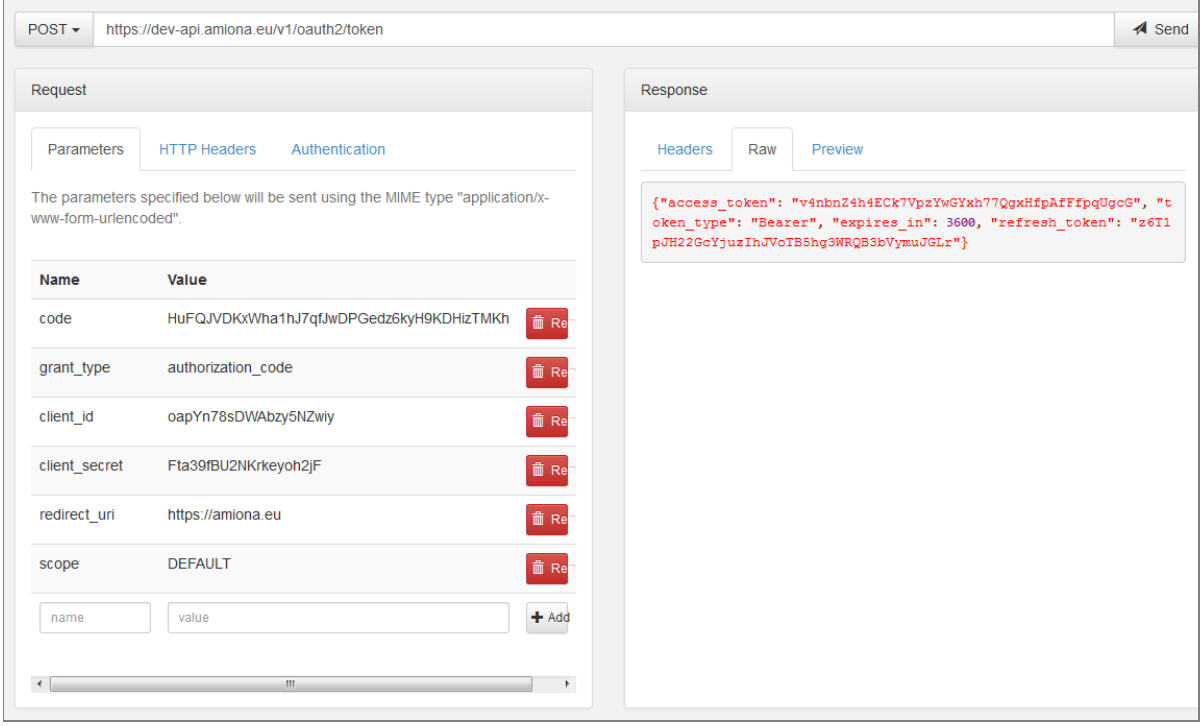
SAMPLE


4.3.1.3 Message [06] & [07]: Exchange authorization code

[06] EXCHANGE AUTHORIZATION CODE			
As the authorization code is only valid once, it has to be exchanged into a more durable access token. However, this token will still have a pre-set lifetime and hence will come with the liability to renew it on a regular basis. Please see section 4.3.2 for further details.			
Directory	`/v1/oauth2/token`	**Method**	`POST ¦ PARAM`
PARAMETERS			
`code`	Authentication code as character string that was included in the redirect_uri.		
`grant_type`	Type of the grant. Should be “authorization_code”.		
`client_id`	Identifier of the 3rd party application for the OAuth server.		
`client_secret`	Secret of the 3rd party application for the OAuth server.		
`redirect_uri`	The URI is pre-set as database entry. However, for additional security this pre-set value has to be confirmed by submitting the same URI as GET parameter.		
`scope`	Scope of functionalities that should be covered by the token; at the moment only “DEFAULT” is available.		
[07] RESPONSE (JSON)			
`access_token`	Token that can be used by the 3rd party application in order to call ACS webservice functions in the name of the user.		
`token_type`	Type of the token. Should be “Bearer” at the moment.		
`expires_in`	Time to the automatic expiration of the token.		

`refresh_token`	Second token that has to be used in order to get a refreshed token after the expiration time.

SAMPLE



4.3.1.4 Further REST Requests

After the 3rd party application has received a valid token, further REST requests can be executed. In order to authenticate the application-user-tuple, the header of each REST requests has to contain an “Authorization” block. The value of this block has to be a concatenation of “Bearer“ (mind the blank at the end) and the BASE64 encoded token (see Figure 18).

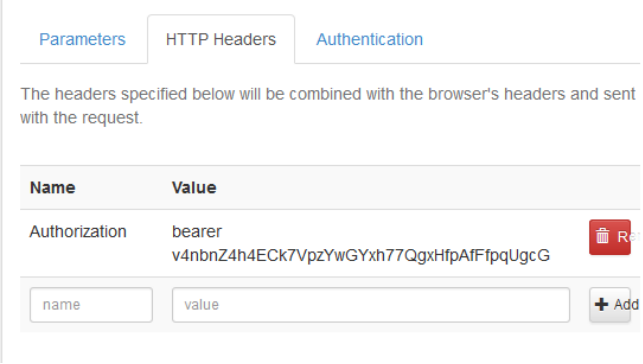


Figure 18: REST authentication via header

Additionally to the header, the respective domain resp. tenant ID has to be included in most REST calls. The basic REST URI consists of the API, the tenant ID, and the service that should be requested.

https:// _API_ / _TENANT_ID_ / {SERVICE}

4.3.2 Refresh token

All user-specific access tokens have a limited life-time. Any call to the ACS system with an expired token will cause a HTTP 401 (Unauthorized) JSON response (see Figure 19). In order to refresh the token again, a POST call to the URL that has initially released the token has to be made.

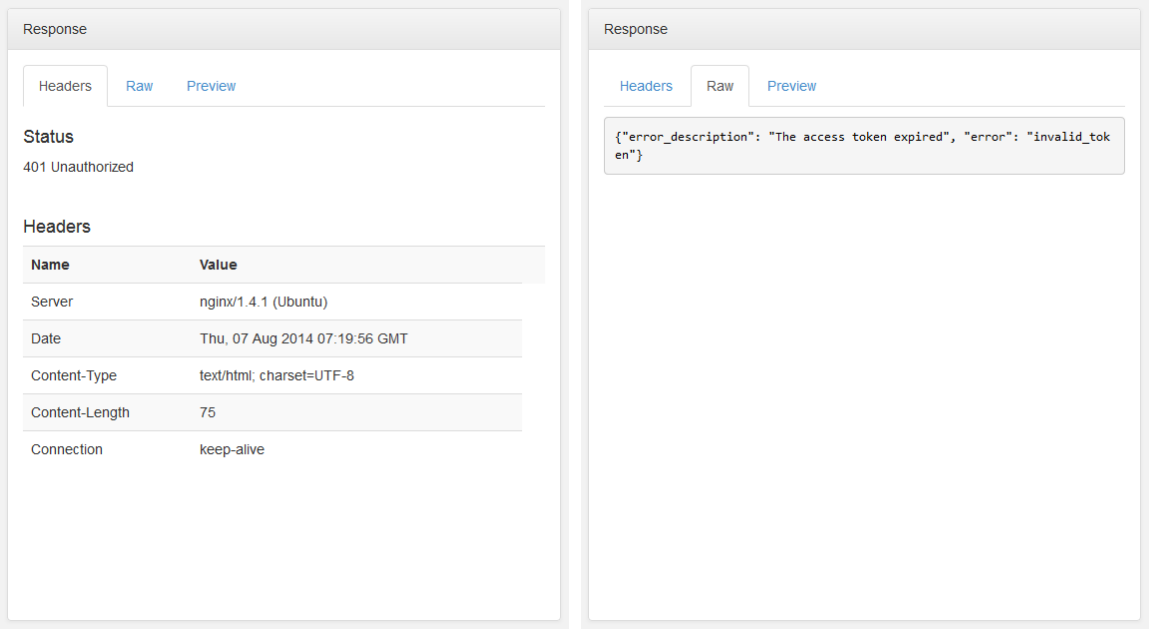
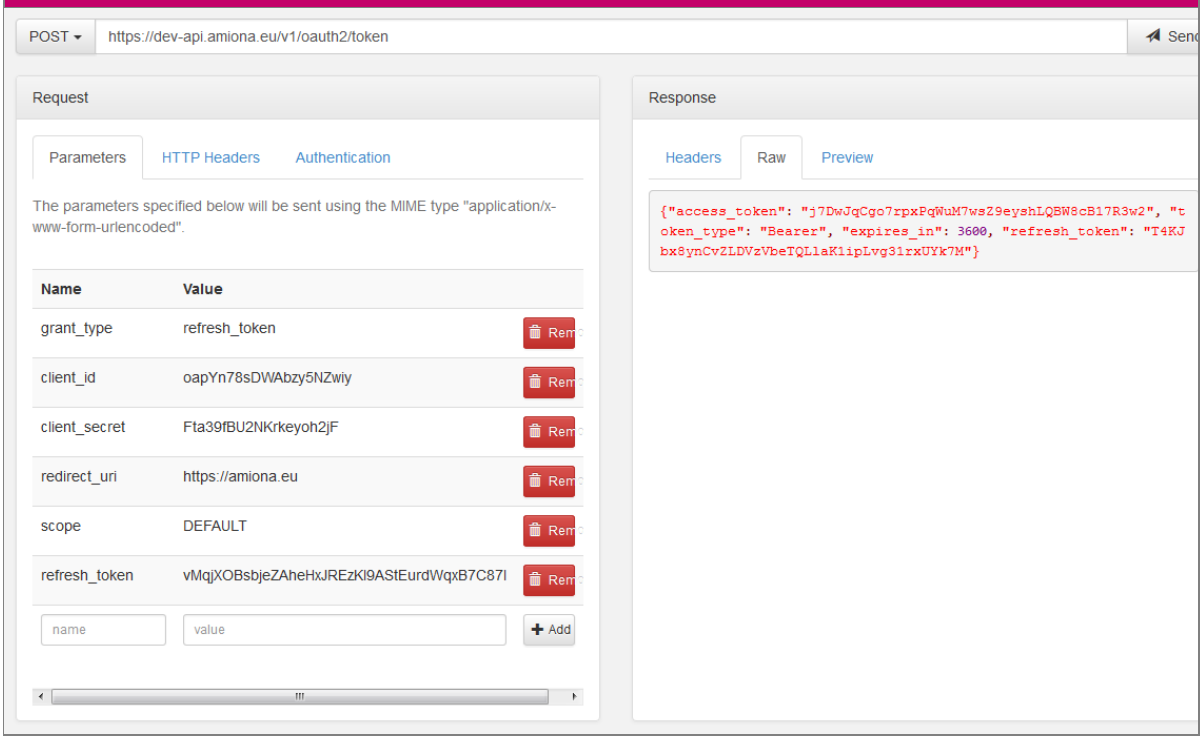


Figure 19: Expired Token Response

REFRESH TOKEN			
Whenever an access token is expired, is has to be renewed in exchange to the refresh_token from the previous request.			
Directory	/v1/oauth2/token	**Method**	POST ¦ PARAM
PARAMETERS			
grant_type	"refresh_token"		
client_id	Identifier of the 3rd party application for the OAuth server.		
client_secret	Secret of the 3rd party application for the OAuth server.		
redirect_uri	The URI is pre-set as database entry. However, for additional security this pre-set value has to be confirmed by submitting the same URI as GET parameter.		
scope	Scope of functionalities that should be covered by the token; at the moment only "DEFAULT" is available.		
refresh_token	The refresh token that was provider together with the access token.		

[07] RESPONSE (JSON)	
access_token	Token that can be used by the 3rd party application in order to call ACS webservice functions in the name of the user.
token_type	Type of the token. Should be “Bearer” at the moment.
expires_in	Time to the automatic expiration of the token.
refresh_token	Second token that has to be used in order to get a refreshed token after the expiration time.

SAMPLE



4.4 ACS Backend Endpoints

This section gives an overview of the most important available endpoints from the ACS backend. More detailed information like, e.g., sample-calls are available upon request. The tenant-id as described in section 4.3.1.4 is simply a synonym for request_user_group_id in the listed endpoints below.

4.4.1 Tenant management

```
>  /create_service_prototype
```

Create a service prototype; the lowest level in the catalogue tree that is not visible as leaf / category field.

```
>  /create_user_group_category
```

Create a category that is linked to a certain user group, e.g., a tenant-id.

```
> /edit_user_group_category
```

Edit a category that is linked to a certain user group, e.g., a tenant-id.

```
> /delete_user_group_category
```

Delete a category that is linked to a certain user group, e.g,. a tenant-id, while considering all possible dependencies.

4.4.2 User authentication

```
> /v1/oauth2/auth
```

Endpoint for starting the OAuth authorization process. Further details can be found in section 4.3.

```
> /{request_user_group_id}/user/{user_id}/check_login/{login}
```

Check whether the submitted login is valid.

4.4.3 Account management

Account management functionalities are grouped into endpoints especially for consumers, providers, performers, share entity services, and user groups.

4.4.3.1 Consumer accounts

```
> /{request_user_group_id}/register_consumer
```

Registration of consumer user accounts.

```
> /{request_user_group_id}/confirm_consumer_registration
```

Submit the confirmation code, so that the user-account is activated.

```
> /{request_user_group_id}/reset_password_request
```

Request a password reset for the user; reset information is sent via e-mail.

```
> /{request_user_group_id}/change_user_password
```

Change the user's password (while user is authenticated).

```
> /{request_user_group_id}/change_user_email_request
```

Change the user's e-mail address and therefore the unique identifier of the user.

```
> /{request_user_group_id}/user/session_data
```

Request the session of the user, containing some basic user-information like name and e-mail.

```
> /{request_user_group_id}/get_consumer_details/{consumer_id}
```

Request the details of the user; this endpoint makes more information available than the session_data endpoint.

```
> /{request_user_group_id}/add_consumer_phone
```

Add a phone number to the consumer account. Remember: all providers have a consumer account, as well.

```
> /{request_user_group_id}/edit_consumer_phone
```

Edit a phone number that is linked to a consumer account.

```
> /{request_user_group_id}/add_consumer_address
```

Add an address to the consumer account. This address can later be used for location selection during the booking process.

```
> /{request_user_group_id}/user/{user_id}/edit_data
```

Edit general user data, e.g., first name, last name, etc.

```
> /{request_user_group_id}/consumers/{consumers_ids}/
```

Returns the details for the submitted list of consumer ids; filter is applied based on the consumer entity.

```
> /{request_user_group_id}/users/{users_ids}/
```

Returns the details for the submitted list of user ids; filter is applied based on the user entity.

```
> /{request_user_group_id}/user/{acs_user_id}/
```

Returns a comprehensive list of the user's details.

```
> /{request_user_group_id}/user/{acs_user_id}/favourites/service/{se
  rvice_id}/
```

Adds the submitted service id to the list of favourite services.

```
> /{request_user_group_id}/user/{acs_user_id}/favourites/services
```

Returns the list of favourite services.

```
> /{request_user_group_id}/user/{acs_user_id}/last_services/{number_
  of_services}/
```

Returns the defined number of services recently used by the user.

```
> /{request_user_group_id}/user/{acs_user_id}/delete
```

Delete the account.

4.4.3.2 Provider accounts

```
> /{request_user_group_id}/register_provider
```

Registration of provider user accounts.

```
> /{request_user_group_id}/confirm_provider_registration
```

Submit the confirmation code so that the provider account is activated.

```
> /{request_user_group_id}/user/{acs_user_id}/transform_account
```

Transform a normal (consumer) user account into a (professional provider) user account.

```
> /{request_user_group_id}/provider/{provider_id}/phone/add
```

Add a phone number to the provider account. Remember: all providers have a consumer account, as well.

```
> /{request_user_group_id}/provider/{provider_id}/address
```

Add an address to the consumer account. This address can later be used for location selection during the booking process.

```
> /{request_user_group_id}/edit_provider_data
```

Edit general provider data, e.g., the company name.

```
> /{request_user_group_id}/provider/{provider_id}/add_performer
```

Add a performer to a provider / company account; the provider itself is the only default performer otherwise.

```
> /{request_user_group_id}/providers/{provider_ids}/
```

Returns the details for the submitted list of provider ids.

```
> /{request_user_group_id}/provider/{provider_id}/addresses
```

Returns the address of the provider; endpoint is also available for consumers.

```
> /{request_user_group_id}/provider/{provider_id}/phones
```

Returns the phone number of the provider; endpoint is also available for consumers.

4.4.3.3 Performer

```
>  /{request_user_group_id}/performer/{performer_id}/activate_account
```

Submit the confirmation code so that the performer account is activated.

```
>  /{request_user_group_id}/performer/{performer_id}/edit
```

Edit general performer data.

```
>  /{request_user_group_id}/performer/{performer_id}/phone/add
```

Add a phone number to the performer account.

```
>  /{request_user_group_id}/performer/{performer_id}/clients
```

Return all clients a performer has been in touch with; endpoint used for direct booking scenarios (autocomplete on existing client list).

```
>  /{request_user_group_id}/provider/{provider_id}/performers
```

Returns all performers that are associated with the submitted provider id.

```
>  {request_user_group_id}/performers/{performer_ids}/
```

Returns the details for the submitted list of performer ids.

```
>  /{request_user_group_id}/get_performers_by_service/{service_id}/
```

Returns all performers associated with the submitted service.

4.4.3.4 Shared entity services

```
>  /{request_user_group_id}/edit_address
```

Edit an address that can be linked to different entities (e.g., user, provider, service).

```
>  /{request_user_group_id}/address/delete
```

Delete an address that can be linked to different entities (e.g., user, provider, service).

```
>  /{request_user_group_id}/edit_phone
```

Edit a phone number that can be linked to different entities; please see also the consumer-specific version of this endpoint.

```
>  /{request_user_group_id}/delete_phone
```

Delete a phone number that can be linked to different entities (e.g,. user, provider).

```
> /{request_user_group_id}/resent_activation_email
```

Resend the activation link e-mail.

4.4.3.5 User Groups

```
> /{request_user_group_id}/user_group/create
```

Create a new user group.

```
> /user_group/{user_group_id}/edit
```

Update the defined user group.

```
> /{request_user_group_id}/user_group/get/{user_group_id}
```

Retrieve the details for the submitted user group id.

```
> /{request_user_group_id}/user_group/get_my
```

Retrieve all user groups associated with the requesting user.

```
> /user_group/{user_group_id}/member/invite
```

Invite a user to join a user group.

```
> /{request_user_group_id}/user_group/get_my_invitations
```

Retrieve all open user group invitations associated with the requesting user.

```
> /user_group/{user_group_id}/member/confirm
```

Confirm the invitation for joining a user group.

```
> /user_group/{user_group_id}/get_invitations
```

Get all invitations for a user group.

```
> /user_group/{user_group_id}/get_members
```

Get all members of the user group.

```
> /{request_user_group_id}/user_group/{user_group_id}/get_potential_
members
```

Get a complete list containing all user that have already accepted the invitation plus the users whose invitation is still pending.

```
> /user_group/{user_group_id}/member/delete
```

Delete a member from a user group.

```
>  delete_user_group', '/user_group/delete
```

Delete the user group.

4.4.4 Service hierarchy

```
>  /{request_user_group_id}/categories/
```

Returns the list of available categories; categories are ordered in a tree-structure that is maintained by a parent attribute.

```
>  /{request_user_group_id}/category/{categories_ids}/
```

Returns the list of details based on submitted category ids.

```
>  /{request_user_group_id}/category/{category_id}/categories
```

Returns a list of all subcategories of the submitted category id.

```
>  /{request_user_group_id}/service_prototype/{service_prototype_id}/
```

Returns the details for a service prototype based on the submitted service prototype id.

```
>  /{request_user_group_id}/category/{category_id}/service_prototypes
```

Returns a list of all service prototypes that are associated with the submitted category id.

```
>  /{request_user_group_id}/search/{pattern}
```

Returns a list of services that correspond to the submitted search pattern.

4.4.5 Service management

```
>  /{request_user_group_id}/provider/{provider_id}/services
```

Returns all services that are associated with the submitted provider id.

```
>  /{request_user_group_id}/create_new_service
```

Create a new service with all related settings (e.g., pricing, direct booking, address restrictions, etc.).

```
>  /{request_user_group_id}/service/is_active/
```

Entity endpoint for Service is_active attribute [POST, PUT, GET].

```
>  /{request_user_group_id}/services/active/
```

Returns all active services (similar to is active).

```
> /{request_user_group_id}/services/{service_ids}/
```

Returns the details for a comma-separated list of service ids.

```
> /{request_user_group_id}/service/{service_id}/bookable
```

Returns all available bookable dates that are associated with the submitted service id.

```
> /{request_user_group_id}/google_service/{service_id}/
```

Returns the details for a service that has been offered by the Google places API.

```
> /{request_user_group_id}/services/random/{number_of_services}/
```

Returns the defined number of random services; can be used, e.g., for a service carousel on a website to demonstrate the scope of available services.

```
> /{request_user_group_id}/services/random/{number_of_services}/{lis
  t_of_rejected_services}/
```

Returns the defined number of random services while excluding a defined list of services by their id.

```
> /{request_user_group_id}/services/special/{number_of_services}/
```

Returns the defined number of random services that are currently offering special conditions (e.g., discounts).

```
> /{request_user_group_id}/services
```

Returns list of services based on several filters (e.g., all services for a catalogue category).

```
> /{request_user_group_id}/get_services_by_performer/{performer_id}
```

Returns all services associated with the submitted performer.

4.4.6 Demand management

```
> /{request_user_group_id}/demand/
```

Create a new demand.

```
> /{request_user_group_id}/service_prototype/{service_prototype_id}/
  demands
```

Retrieve all open demands that are associated with the submitted service prototype id.

```
> /{request_user_group_id}/demand/{demand_id}
```

Retrieve the details for a demand by its id.

```
> /{request_user_group_id}/demand/is_active/
```

Retrieve and update the is_active field for demands.

```
> /{request_user_group_id}/demand/edit/
```

Update the details of a demand.

```
> /{request_user_group_id}/demand/get_users_demands/
```

Retrieve all demands associated with the requesting user.

4.4.7 Appointment negotiation

```
> /{request_user_group_id}/appointment/request
```

Request an appointment for a certain service [registered users only].

```
> /{request_user_group_id}/appointment/appointment_request_without_registration
```

Request an appointment for a certain service [unregistered users only].

```
> /{request_user_group_id}/appointment/appointment_request_with_registration
```

Request an appointment for a certain service [unregistered users only; but user account is created during request].

```
> /{request_user_group_id}/appointment/ad_hoc_request
```

Request an appointment for a placeholder service and a provider not yet registered to the system.

```
> /{request_user_group_id}/appointment/{appointment_id}/cancel
```

Cancel an appointment request or an appointment that has been confirmed before.

```
> /{request_user_group_id}/appointment/{appointment_id}/dispatch
```

Assign an appointment request to an employee / to a performer [only available for providers].

```
> /{request_user_group_id}/appointment/{appointment_id}/refuse_internal
```

Reject appointment request dispatching and send the request back to the provider [only available for performers].

```
> /{request_user_group_id}/appointment/{appointment_id}/refuse_final
```

Definite refuse of an appointment request (in contrast to dispatch and refuse internal an action that the consumer will be notified about).

```
> /{request_user_group_id}/appointment/{appointment_id}/accept
```

Accept an appointment request or reschedule.

```
> /{request_user_group_id}/appointment/{appointment_id}/reschedule
```

Reschedule an appointment from both sides, consumer and provider / performer.

```
> /{request_user_group_id}/appointment/{appointment_id}/alternative_
  date_propose
```

Propose multiple alternative dates [only available for the performer and only during the first response].

```
> /{request_user_group_id}/appointment/{appointment_id}/alternative_
  date_accept
```

Accept one of the proposed alternative dates.

```
> /{request_user_group_id}/appointment/{appointment_id}/executed
```

Mark an appointment as executed (and submit deviating financial or expense values if needed).

```
> /{request_user_group_id}/appointment/{appointment_id}/comment
```

Add a comment to an appointment.

```
> /{request_user_group_id}/user/{acs_user_id}/appointments
```

Returns a list of all appointments of the user.

```
> /{request_user_group_id}/user/{acs_user_id}/get_filter_information
```

Returns information for the filter sets that are available (filters like 'today' or 'action needed'). Returned information set contains the number of related appointments for each filter.

```
> /{request_user_group_id}/service/{service_id}/appointments
```

Returns all appointments that are associated with a service id.

```
> /{request_user_group_id}/user/{acs_user_id}/appointments/{year}/{m
  onth}/
```

Return all appointments that are associated with the submitted month for the requested user.

```
> /{request_user_group_id}/user/{acs_user_id}/appointment/{appointme
  nt_id}/
```

Get user-specific information set of an appointment by its id; the returned set of information might differ with respect to the user role (consumer, provider, etc.).

```
>  /{request_user_group_id}/user/{acs_user_id}/appointments/open/
```

Returns a user's set of open appointments (appointments in negotiation and not marked as executed); used, e.g., for the delete action of a user.

```
>  /{request_user_group_id}/user/{acs_user_id}/mark_appointment_as_se
   en
```

Marks an appointment as seen; information is used for the 'unseen' flag / filter.

```
>  /{request_user_group_id}/bookable/
```

Create a bookable date entry.

```
>  /{request_user_group_id}/bookable/{id}
```

Retrieve the details for a bookable date and / or update the bookable date.

```
>  /{request_user_group_id}/get_provider_bookables_date/{provider_id}
```

Retrieve all bookable dates for the submitted provider id.

```
>  /{request_user_group_id}/get_service_bookable_dates/{service_id}
```

Retrieve all bookable dates associated with the submitted service id.

```
>  /{request_user_group_id}/service_bookable_date/
```

Create association between a bookable date and a service.

```
>  /{request_user_group_id}/service_bookable_date/{service_bookable_d
   ate_id}
```

Retrieve the details for the association between a service and a bookable date.

4.4.8 Support functions

```
>  /{request_user_group_id}/ics/{acs_user_id}/{appointment_id}
```

Returns the ICS file for the submitted appointment id.

```
>  /{request_user_group_id}/ics/subscribe/user/{acs_user_id}/{hash}
```

Returns the whole calendar feed for the submitted user id.

```
>  /{request_user_group_id}/ics/subscribe/provider/{acs_provider_id}/
   {hash}
```

Returns the whole calendar feed for the submitted provider id (includes a joint calendar for all performers).

```
> /{request_user_group_id}/calendar/{acs_user_id}/get_calendars
```

Returns a list of associated calendar feeds.

```
> /{request_user_group_id}/calendar/{acs_user_id}/add_calendar
```

Adds a calendar feed for the user.

```
> /{request_user_group_id}/calendar/{acs_user_id}/get_calendar/{cale
ndar_id}
```

Returns the pre-parsed feed of the external calendar.

```
> /{request_user_group_id}/calendar/{acs_user_id}/update_calendar/{c
alendar_id}
```

Updates the credentials for the external calendar.

```
> /{request_user_group_id}/calendar/{acs_user_id}/delete_calendar/{c
alendar_id}
```

Deletes the external calendar.

```
> /{request_user_group_id}/content/{key}
```

Retrieve pre-defined content (e.g,. templates that can be used for presentation purposes).

```
> /{request_user_group_id}/content/{key}/{lang}
```

Retrieve pre-defined content for a specific language.

```
> /{request_user_group_id}/image
```

Upload endpoint for images (e.g., profile image, service image, etc.).

```
> /v1/oauth2/intermediary_login
```

Allows an intermediary to login to any other account in the system.

```
> /{request_user_group_id}/get_available_currencies
```

Returns all available currencies.

```
> /{request_user_group_id}/get_available_units
```

Returns all available units.

```
> /{request_user_group_id}/prices/{price_ids}/
```

Returns the details for prices associated with the submitted price ids.

5. TASKS OVERVIEW

This section gives a brief overview of the sub-tasks that have been defined in the project's proposal for task 3.4 and deliverable D3.3. All sub-tasks are briefly described and results are summarized.

5.1 Development of modules for accessing (external) information sources

Besides the service infrastructure, GeTVivid aims at integrating additional information sources like news or schedules. These sources, however, need to be adjustable to local peculiarities (e.g., region-specific news and weather forecasts).

The development of modules for accessing external information sources has been postponed until the mid-term review. Further investigation has identified that a backend integration is not expedient, which is why this task has been excluded from the service platform. Additionally, the development of a broader functional scope has been emphasized.

5.2 Business process integration

The integration of a business process logic aims at two aspects. First, the application of a process logic is advisable in order to handle different scenarios (e.g., informal vs. professional services) while maintaining a minimum of flexibility for adjustments. Second, the overall architecture of the used process engine enables a seamless integration with professional service providers that is expected to become essential for larger companies (see also results for task 5.2).

A business process engine has been developed and integrated into the backend-layer. Technically, the process engine consists of interfaces with an appropriate processing logic that allows to run modelled processes and to call and react to hook-points. All request or booking process variants have been realized based on this process engine. Additionally, an Enterprise Resource Planning (ERP) system integration has been realized as proof of concept. Further insights and requirements are expected to be derived from the field-trials.

5.3 Management of user profiles, interfaces for TV and mobile clients

One of ACS's backend related functionalities is the user management system, which is integrated deeply as almost all actions respectively requests are associated with a certain user or company (e.g., adding a service, requesting a service, etc.). Due to the underlying safety policy of ACS, however, access to user details like the address is restricted to the respective user and only opened for others due to appointment coordination related requirements (e.g., in order to keep an appointment at the client's location the address has to be shared with the provider). The user system with its profiles has been developed based on OAuth 2.0 and is therefore open for other components of GeTVivid (see section 3 and especially 4.3).

In order to support HbbTV and mobile device related clients, all backend functionalities have been extended with REST web-services (please see section 3 for further details).

5.4 Broadcast playout centre coordination and interfacing

The broadcast playout centre has initially been designed to make information available solely via broadcast channels. Due to technical restrictions with respect to offline respectively broadcast content delivery and especially due to legal limitations, a broadband-based approach has been used. The major problem for this task is the channel-content sovereignty of broadcasting companies. This would mean, that for each channel resp. for each broadcasting company a bilateral contract would be required. Hence, a broadband approach is more realistic and achievable.

5.5 Development and deployment of the service provider frontend

A web-based frontend for professional service providers has been developed and is available for the field-trials. For acceptance reasons, professional provider's responses to appointment requests are not only possible via a login-system, but as well by direct-login links that are sent with each request and update notification. Technically, (informal) users would be able to use the same frontend, as well. Due to the projects alignment, however, they are intentionally excluded and limited to the special HbbTV and mobile UI.

5.6 Profiler Module

The following sections describe the Profiler Module architecture and how the module exposed a set of web services in order to establish a communication channel between the GeTVivid platform and the Profiler Module.

A profile is a description of someone containing the most important or interesting facts about him or her. In the context of users in software applications, a user profile or user model contains essential information about an individual user. The motivation of building the user profiles is that users differ in their preferences, interest, background and goals when using software applications. Discovering these differences is vital in order to provide personalized services to the users (Intelligent User Profiling).

In order to provide personalized services for each user, a machine learning technique has to be applied due to the fact that the software needs to learn through the interactions between the user and the platform. User profiling implies inferring unobservable information about users from observable information about them, that is, their actions or interactions (Artificial Intelligence).

5.6.1 Profiler architecture

The current architecture of the Profiler has totally changed. Previously, three main components of the Profiler were developed:

- **GeTVivid Simulator**: it was an ACS simulator which was the data provider and it also tracked the user's behaviour. The aim of the GeTVivid Simulator was to enable a user interface which emulated the GeTVivid platform for testing purposes related to the Profiling Module.
- **Profiler Client**: it was a way to create an information bridge between the GeTVivid Simulator and the Profiler. The Profiler Client knew how to connect with the Profiler Module and it is in charge of sending

the user's data to the Profiler Module. When the data sent to the Profiler has been analyzed, the Profiler Client returns the results to the GeTVivid Simulator. The Profiler Client worked as a bridge between the GeTVivid Simulator and the Profiler Module. The Profiler Client also did the datamining process.

- **Profiler Module**: it is the core of the data mining process, the Profiler is the place where all the machine learning tasks are executed. The Profiler took the information from the Profiler Client and return a user's suggestion based on the analysis of the data sent by the Profiler Client.

A brief view of the old architecture is shown in the Figure 20: Old profiler architecture.

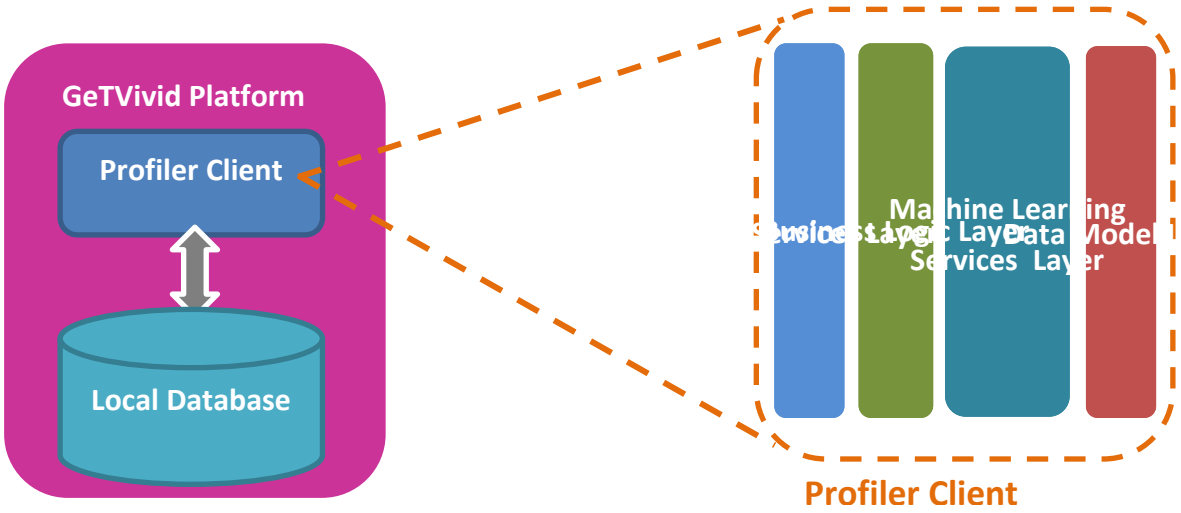


Figure 20: Old profiler architecture

The old architecture has evolved based on the GeTVivid platform requirements. Firstly, the Profiler Client was the responsible of the datamining task because it has direct access to the GeTVivid platform database through an SSH connection. This situation caused a high computational costs and it also increase the response time of the GeTVivid platform, so if the GeTVivid application will run in either a Smart-TV or in mobile devices the computational cost and the response time needed to be reduced.

Currently there is only one component, the Profiler API. The Profiler now is composed by an API which can be accessed through a set of web services using JSON as the exchange information format. A brief example of the architecture is shown in the Figure 21: Current profiler architecture. With this architecture the computational extra cost and the extra response time is solved due to the fact that the Profiler API is in charge of the datamining process, therefore the different GeTVivid clients such as the smart-TV or mobiles devices will be freed of the datamining tasks.

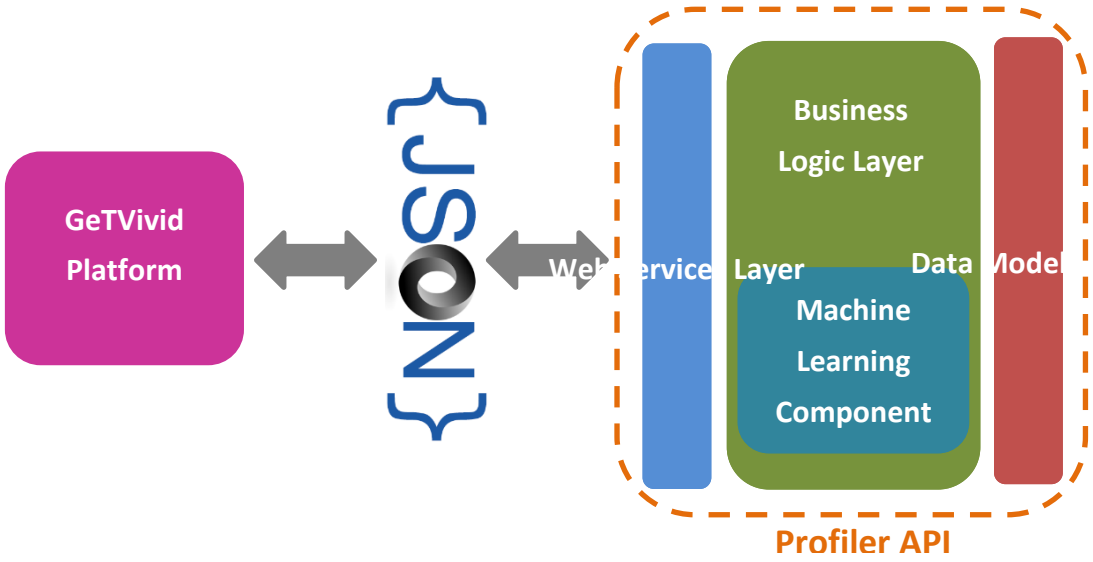


Figure 21: Current profiler architecture

The aim of the Profiler API is to ensure a good suggestion service to the user based on his behaviour when the user is using the platform. The ACS tracks the user interactions with the platform and store some user actions such as purchased offers or demands which were visited by the user. With all the gathered data the Machine Learning Services give a suggestion to the user.

5.6.2 Business logic layer

Is the part of the program that encodes the real-world business rules that determine how data can be created, displayed, stored, and changed. The business are the development of the system's requirements.

Machine learning component

One important piece of this business layer is the Machine Learning Component which is in charge of the data mining tasks. The Machine Learning Layer is composed by two main components, as it is shown in the Figure21:

- Data Pre-processing layer.
- Data Mining Algorithms layer.

Both main components are described in the next sections.

- **Data pre-processing layer**

  Before giving a suggestion to the user, the Data Pre-processing layer does an important task by using different techniques of data pre-processing. The Data Pre-processing phase is necessary due to the fact that most of the suggestions provided by the Profiler Module are based on processing natural language.

  The importance of pre-processing is emphasized by the fact that the quantity of training data grows exponentially with the dimension of the input space. It has already been proven that the time spent on

pre-processing can take from 50% up to 80% of the entire classification process (The Mining Mart Approach to Knowledge Discovery in Databases).

The Profiler Module pre-processing layer is one of the most important parts of the module because is the responsible of applying different pre-processing techniques such as Stop-words filtering or Tokenizers among others before using any kind of machine learning algorithm. A summarized structure of this pre-processing layer is shown in the Figure 22: Data pre-processing layer.

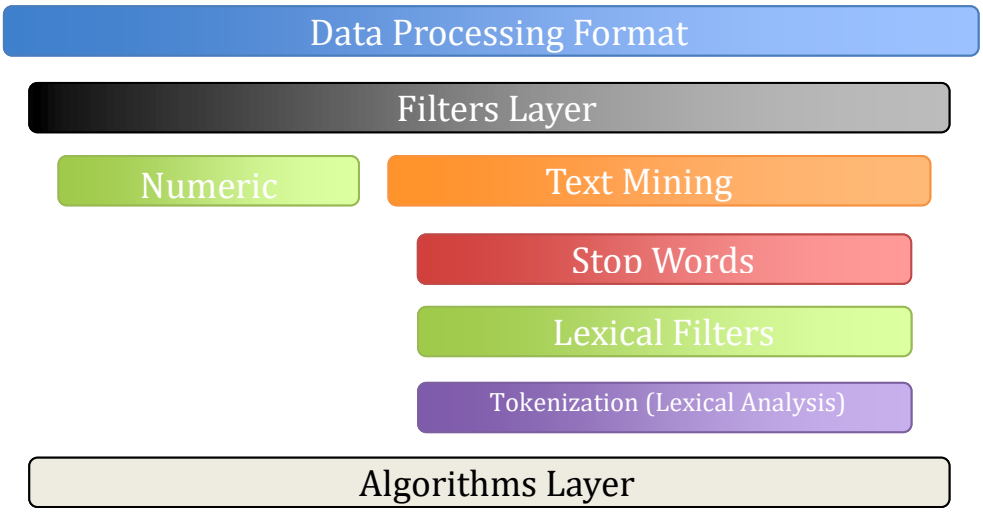


Figure 22: Data pre-processing layer

The Figure 22 shows a high-level view of the pre-processing layer with some of the most important pre-processing techniques such as (Pre-processing Techniques in Text categorization):

- **Filters layer:** in this layer the Profiler Module selects the features which are going to be analysed based on threshold techniques like Term Frequency – Inverse Document Frequency.
- **Stop words:** in natural language there are a set of words which don't give any semantic information and thus they can be omitted in the text categorization tasks in order to improve the performance and the accuracy of the results.
- **Lexical filters:** taking into account the domain of the GeTVivid Profiler Module, there are many filters that can be used so as to reduce the number of selected features and also optimize the performance of the text categorization algorithms. For example, the Stemming technique gets the stem or root of a word, e.g., *"computing", "computer"* and *"computers"* when the stemmer filter is applied, the 3 words will be transform into its root: "comput".
- **Tokenization**: is the process of breaking a stream of text into words, phrases, symbols or other meaningful elements called tokens. This set of token will be used as the input for the text categorization algorithms.

Once all the pre-processing phase is done, all the pre-processed data is the input for a data machine learning algorithm. The next section will introduce the selected machine learning algorithms used in the Profiler Module.

- **Machine learning algorithms**

  The machine learning algorithms which compounds the Machine Learning Component have been selected based on different studies and researches on the subject (A Review of Machine Learning Algorithms for Text-Documents Classification, A comparative Study on Different Types of Approaches to Text Categorization, Text Categorization and Machine Learning Methods: Current State of the Art). In this researches, there is a common thought about some types of algorithms and how they work categorizing natural language. The Profiler Module take advantage of two kind of the following algorithms:

  - **K-nearest neighbour (K-NN)**: The k-nearest neighbour is used to test the degree of similarity between documents and k training data and store certain amount of classification data, thereby determining the category of test documents. Generally, the K-NN method performs well even when it needs to handle classification tasks with multi-categorized data. One disadvantage of the K-NN is that it uses all the features in the classification tasks and this behaviour causes a high computational cost when it handle datasets which will be growing continuously (A Review of Machine Learning Algorithms for Text-Documents Classification, KNN with TF-IDF Based Framework for Text Categorization).
  - **Decision trees**: The main advantage of the decision tress is their simplicity in understanding and interpreting the results even for non-experts users. If the Decision Tree has to handle with a small number of structured attributes its performances, understandability and simplicity are all advantages. So in order to use a Decision Tree in the GeTVivid platform needs a very good pre-processing phase so that a small set of features will be used in the final text categorization (A comparative Study on Different Types of Approaches to Text Categorization).

5.6.3 Web services layer

The Profiling module exposes a set of web services which can be consumed by the GeTVivid platform in order to get the users' suggestions. The Rest API exposes a resource named "profiler" with GET and POST HTTP metohds, receiving as input a JSON document.

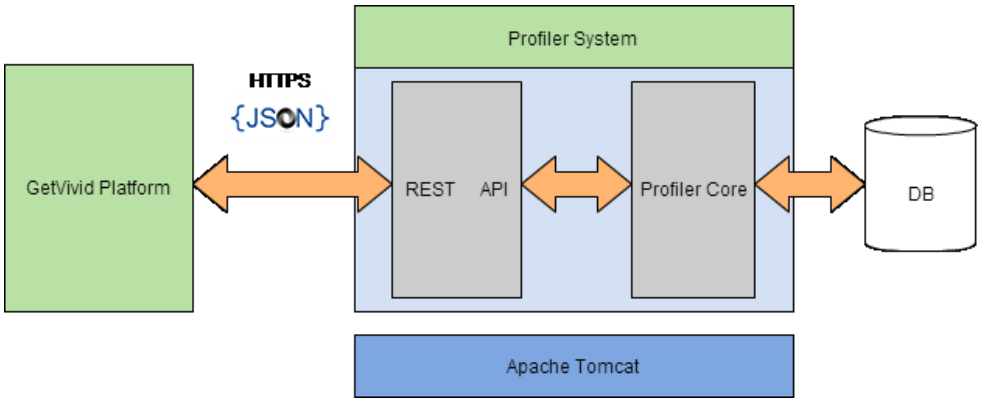


Figure 23: Web services layer architecture

The Table 2 shows HTTP methods, which are commonly used in REST architectures, already developed in the Web Services Layer.

Uniform Resource Identifier (URI)	GET	POST
http://iat.isoin.es/profilerAPI-web/api/resourceParent/resource	The HTTP GET method is used to read or retrieve a representation of a resource.	The HTTP POST method is used to create new resources.

Table 2. HTTP methods for REST services.

The GeTVivid platform only has to know the Uniform Resource Identifier of each Profiler web service in order to access to its resources. When the Profiler server notice that someone is calling, it automatically uses this URI in order to identify the resource requested by the client, in the GeTVivid system the client is the GeTVivid platform. The API will give a response to the call and in the response the client will get the requested resource or if an error has occurred, the API includes an error code as it is shown Table 3.

Code	Meaning
0	Everything is fine.
1	Authorization error.
2	Unexpected Internal Error.
3	Unknown service.
4	Service not available.

Table 3. Profiler API REST errors.

A more detailed explanation for each web service can be seen in the following sections.

Badges suggestion service

The GeTVivid platform introduces a gamification concept based on the assignation of a set of badges to each user based on a set of interaction with the platform, such as buy more than 10 offers which belong to the shopping category, in that case the GeTVivid platform will assign a Shopping badge : “Shopping Junior” to the user.

Figure 24: Badges shopping example

The Profiler Module will use a set of keywords for each badge in order to classify the user's behaviour into one of the gadgets categories. The GeTVivid platform has to send to the Profiler Module a call for the Badges service so that a badge can be assigned to a user.

Category suggestion service

The category suggestion service will suggest an offer or demand category based on its K nearest neighbours based on the *Title* attribute of the offer or demand. This service will help the user when he/she is creating a new offer or demand. Once the user wants to create a new offer or demands, the user has to fill a formular with all the related offer or demand data. Once the user has filled the "Title" field of the formular, the service will set the category of the current offer or demand on the formular automatically.

Offer and Demand suggestion service

The Offer and Demand suggestion service suggests a list of offer and demands which are suitable for the user. All these offers and demands returned by the service are selected by taking into account the user's search history and its current search in the GeTVivid platform and thus some relevant offers and demands are suggested to the user. For example, if a user is looking for a gardening offer and in the search bar the user introduces: "Trees Pruning", the Offer and Demand suggestion service will return a set of services based on this search and also some other services related with his before searches on the platform.

5.6.4 Data model layer

In order to suggest offers or demands to the users, a conceptual understanding of the user has to be developed in the Profiler Module. With this user modelling, the Profiler Module creates the bases for the suggestion services because the user modelling represents the interactions between the user and the platform which are going to be used as an input for the Web Services Layer methods.

The Data Model Layer also handle the information exchange between the data sent by the GeTVivid platform and the Business Logic Layer, in other words, is in charge of transform the information sent by the GeTVivid platform in JSON format to an understandable Business Logic Layer object.

6. OUTLOOK AND FURTHER DEVELOPMENT

During the GeTVivid project, ACS has been used as backend for a broad range of functionalities. However, against the background of general applicability and extensibility of the backend system (for the use case of appointment coordination), some design decision and functionalities have not been introduced directly into the backend, but are covered by a middleware layer instead. With consideration of the field trial results and the developed business model(s), the functionalities should be examined again and evaluated due to their contribution for potential exploitation strategies. Where a significant contribution is given, the (re-) integration into a joint backend should be aspired. Two functionalities respectively aspects that might be considered are:

- Request Chains: Some functionalities require a chain of requests that is executed (e.g., the retrieval of recursive structures like the catalogue, or the creation and maintenance of direct bookable dates for service offers). At least some of these requests might be subject to simplification thereby disengaging the middleware layer from handling such chains.
- Message Exchange: ACS offers functionalities for appointment coordination related message exchange. Message that are sent outside of the scope of appointment coordination (i.e., chats) do not represent an integrated functionality at the moment. As such use cases have been assessed crucial, the integration into ACS might be useful.

Additionally to this rather optimization related outlook, further development can facilitate exploitation strategies. One the one hand, most of the developed functionalities are available for consumers via mobile app and TV client; the web-interface available for providers covers primarily the limited use cases that have been identified. By enhancing the web-interface with some of the consumer-specific functionalities (e.g., demand-driven appointment coordination), a broader interest group and additional use cases might be addressable. On the other hand, the provider integration can be extended by additional channels (e.g., messenger or phone). Besides contributing to providers' acceptance in general, different means of communication may be appropriate for the acceleration of the coordination process in general.

REFERENCES

Armour, F. J., Kaisler, S. H., & Liu, S. Y. (1999). A Big-Picture Look at Enterprise Architectures. *IT Professional*, *1*, 35–42.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley Professional.

Garlan, D., & Shaw, M. (1993). An Introduction to Software Architecture. *Advances in Software Engineering and Knowledge Engineering*, *1*(January), 1–40. doi:10.1142/9789812798039_0001

Gorton, I. (2011). *Essential Software Architecture (2. ed.)*.

IEEE-SA Standards Board. (2000). IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE Std*, *1471-2000*, 1–23. doi:10.1109/IEEESTD.2000.91944

Maciaszek, L. A. (2006). From Hubs Via Holons to an Adaptive Meta-Architecture - tlie ' AD-HOC ' Approach, *227*, 1–13.

Maciaszek, L. A., Skalniak, T., & Biziel, G. (2014). A Meta-architecture for Service-oriented Systems and Applications. *Proceedings of the Fourth International Symposium on Business Modeling and Software Design*, 20–28. doi:10.5220/0005423900200028

Smeda, A., Khammaci, T., & Oussalah, M. (2005). Meta Architecting : Towards a New Generation of Architecture Description Languages. *Journal of Computer Science*, *1*(4), 454–460. http://www.doaj.org/doaj?func=abstract&id=1099285