D4a.2 Smart Citizen Assistant Software Architecture

Final | 1.0 | 30.09.2014

Verantwortlicher Partner:	Siemens AG Österreich
Autoren:	Josef Aigner Stefan Bischof Deepak Dhungana Roman Tobler
Work package partner	Siemens, ETA, Wien 3420, Energiecomfort, Wiener Netze, MA18, MA20, MA21, ÖIR

Executive Summary

Open access to large amounts of data from cities that are also dealing with booming numbers of residents will be critical for smart cities to adequately address the challenges that will likely arise. By making data available in a city to its residents, city administrators and businesses, it is possible to open up a new market for innovative use of data. In order to make this possible, APIs (Application Programming Interfaces) are implemented, which are controlled channels for the dissemination of data. API management is the process of publishing, promoting and overseeing application programming interfaces (APIs) in a secure, scalable environment. It also includes the creation of end user support resources that define and document the API. The goal of API management is to allow an organization that publishes an API to monitor the interface's lifecycle and make sure the needs of developers and applications using the API are being met. This document describes an architectural blueprint for a smart citizen assistant, which is an interface to data available and collected in a smart city.

The architecture is designed such that the involved stakeholders (data publishers, application developers and data consumers) interact with the smart citizen assistant in a controlled manner. Workflows are defined to add new data to the database, to make selected datasets available to the public through defined interfaces and for the management of users, rights and responsibilities. After the smart citizen assistant and the data access services are deployed in a server, data-owners will have the possibility to publish their data in a controlled manner. Developers will have the possibility to design and implement innovative applications (APPs) using the available data and the citizens will profit from using the applications. This is therefore a win-win situation where the data publishers, developers and citizens can profit by forming new business partnerships, information campaigns, or public services. However, from a legal perspective, the issues around data privacy, data ownership and protection of personal data remain open:

- The publisher of the data must ensure that the data made available through the API do not violate existing laws and regulations governing data privacy.
- The developer of the Apps must ensure that the data consumed through the API is used for legitimate reasons respecting the users' privacy requirements.

Smart citizen assistant is a software component which needs resources for operation and maintenance. In order to make this component available after the runtime of the project, resources must be assigned and allocated for continuous operation, maintenance, updates and upgrades. This document describes in detail, how the internal components of an API management component work together to achieve a set of higher level goals and how the integrity and privacy requirements are considered in all the interactions.

List of Figures

Abbreviations

API	Application Programming Interface
ASCR	Aspern Smart City Research
BI	Business Intelligence
CSV	Comma Separated Values
DB	Database
DC	Data Concentrator
JSON	JavaScript Object Notation
KB	Knowledge Base
MDM	Meter Data Management
MDMS	Meter Data Management System
MQ	Message Queue
PLC	Programmable logic controller
ReST	Representational State Transfer
SCA	Smart Citizen Assistant
SG	Smart Grid
SUL	Smart Urban Lab
UDA	Unified Data Architecture
UI	User Interface
XML	Extensible Markup Language

Contents

1 Introduction

Transform+ is a research project funded by „Klima- und Energiefonds“, as part of the Austrian research funding agency FFG. The goal of the project is to prepare and operationalize the contributions of the EU project „TRANSFORM“[1] in the city of Vienna. One of the outputs of the project will be implementation plans for pilot projects in "Liesing-Groß Erlaa" and "aspern Seestadt" (Smart Urban Labs).

As part of this project, a "Smart Citizen Assistant" is to be designed for aspern Seestadt, which is a platform for making smart city data available to the residents. The aim is to provide a novel interface through which relevant data can be accessed individually and in a timely manner.

1.1 Project Status

As planned in the project proposal for Transform+, the work package 4A is working according to the planned schedule. As it can be seen in Figure 1, this deliverable is the second deliverable of the work package, where the technical solution of the smart citizen assistant is described. The previous deliverable is available online[2]. The next deliverable, due May 2015, will be an implementation of the solution concepts presented in this document.

Meilensteine, Ergebnisse und Deliverables:	
Meilensteine	
M4a1	Start der Smart Citizen Assistant Implementierung (05/2015)
M4a2	Smart Citizen Assistant Prototype – Abschluss (11/2015)
Deliverables	
D4a.1	Requirements Analyse und Szenariendefinition: Anforderungsanalyse und Beschreibung des Standes-der-Technik und deren Anwendbarkeit beim Entwurf des Smart Citizen Assistant (Dokumentation) (02/2014)
D4a.2	Architektur und Design der Smart Citizen Assistant Schnittstelle: Definition und Beschreibung des Smart Citizen Assistant Prototype, grundlegende Design-Entscheidungen (Dokumentation) (09/2014)
D4a.3	Implementierung: Prototypische Implementierung der Schnittstelle (Software) (05/2015)
D4a.4	Evaluierungsergebnisse der Testphase: Beschreibung der System-Architektur und der beispielhaften Verwendung der Schnittstelle durch eine Pilot-Applikation (Dokumentation) (11/2015)

Figure 1. An excerpt of the planned deliverables of the project (copied from the project proposal).

[1] TRANSFORM: Transformation agenda for low carbon cities http://urbantransform.eu/
[2] Transform+ website http://www.transform-plus.at/

1.2 Purpose of this document

This document provides the basis for implementing the smart citizen assistant, by defining the system architecture, the key functionality of the API manager, the key methods of the API itself and the roles of different stakeholders involved in the process. With this architectural blueprint, it is possible to verify that the smart citizen assistant fulfills its stakeholders' needs prior to building it. This represents substantial cost-saving and risk-mitigation.

Concrete implementation of the smart citizen assistant is out of scope of this document and will be considered for implementation *as the next step in* the project. The choice of concrete technology, programming languages and libraries is also out of scope. The components described in this document simply reflect the functionality required by the final implementation.

1.3 Contents of this document

This document is structured as follows: In Chapter 2, we describe the architectural goals of the smart citizen assistant and the constraints that must be considered during implementation and deployment. Chapter 3 is about the functional building blocks of the architecture, describing the different components required. Chapter 4 describes the dynamics of the smart citizen assistant in the form of workflows. Chapter 5 sketches the different deployment scenarios and the dependencies to other projects and data sources. Chapter 6 describes example APIs and APPs to demonstrate the added value of the smart citizen assistant – enabling integration of multiple data sources in the context of a smart city.

2 Architectural Goals

Software architecture is a blueprint for implementing the software and it provides a basis for analysis of software systems' behavior before the system has been built. Documenting software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects. This chapter describes the goals and constraints in designing the smart citizen assistant architecture.

The smart citizen assistant has to consider the required functionality of all of the involved stakeholders and the flow of data through the system. At the same time, the design must also consider quality attributes such as fault-tolerance, backward compatibility, extensibility, reliability, maintainability, availability, security, usability etc. Stakeholder concerns often translate into requirements on these quality attributes, which are variously called non-functional requirements, extra-functional requirements, system quality requirements or constraints. The architecture of the smart citizen considers these non-functional attributes and constraints. Similarly, the smart citizen assistant has to cater to a variety of stakeholders such as data owners and publishers, end-users and application developers. These stakeholders all have their own concerns with respect to the system.

Balancing these concerns and demonstrating how they are addressed is part of designing the system. The architecture described in this document also represents an overall vision of what the smart citizen should and how it should do it. This vision may deviate from its implementation. However, this architecture document acts as a reference, making sure that additions to the system are in line with the architecture, hence preserving conceptual integrity.

2.1 Stakeholders

Any person, group or organization that has interest or concern in data in the context of a smart city is a valid stakeholder for the smart citizen assistant. The primary stakeholders are direct users of the platform, or the applications that build upon the platform (For example data publishers, application developers and application users). The secondary stakeholders do not engage in direct interaction with the platform, but maybe affected or can affect the operation of the platform (For example city administration, non-profit organizations, or the media). In this section, we describe the interest, roles and responsibilities of the primary stakeholders.

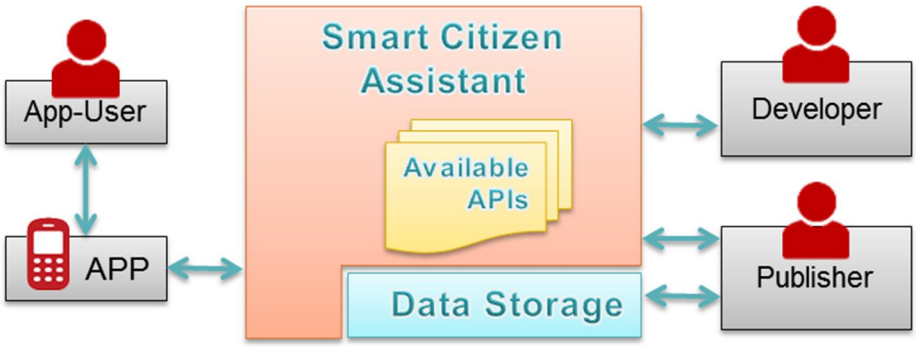

Figure 2 Overview of primary stakeholders and interaction with the smart citizen assistant

2.1.1 Data Publisher

Any person or organization, who owns certain data and wishes to make these available to the public can do this through the smart citizen assistant and thereby takes the role of a data publisher. The publisher is responsible for making sure that it is legitimate to make the data available to the outside world. The smart citizen assistant supports this process by providing tools to streamline publishing, sharing, finding and using data. The smart citizen assistant is aimed at data publishers (national and regional governments, companies and organizations) wanting to make their data open and available. The publisher provides data through APIs and services to developers and eventually to users. Depending on the type of data, there can be two modes of publishing data.

- Personal data refers to data about an individual who can be identified from that data; or from that data and other information to which the organization has or is likely to have access. Such data can only be used or transferred with the individual's knowledge and consent. This means, if such data is made available through the smart citizen assistant, the end-user is required to login to the system to access her personal data (e.g., smart meter data, personal location tracking data etc.).
- General data refers to data that does not allow direct or indirect association of the data to an individual. Such data is typically publicly available and there are no privacy issues that must be considered, when publishing such data. Nevertheless, security measures must be implemented in the smart citizen assistant to ensure controlled access to data.

2.1.2 Application Developer

A data platform provides data for multiple applications through predefined interfaces (APIs) and standards. A developer uses data provided by APIs (and their service implementations) to create a consumer (e.g., an app, web application, or desktop application) which is used by the user. This enables open and user driven innovation for experimenting and validating future data-driven businesses and services. The role of a developer can be taken both by individuals and organizations.

By being able to access the data through APIs, where key tasks such as user management, privacy management are taken care of by a central platform, the developer is able to speed time to market through simplified access to data via mobile-optimized APIs, and comprehensive real-time mobile analytics to power user engagement and measure success. This means, the use of API via applications is monitored and made available to the developers to secure end-to-end visibility into test coverage, API usage, user experience, app adoption and performance.

As the APIs are published through existing standard technologies, the developer is free in choosing the mobile platform (e.g., Apple IOS, Android, Windows Phone, Blackberry etc.) or even desktop application runtime environments (Apple, Windows, Linux). Access to data should not be restricted to a certain platform, programming language or runtime environment.

2.1.3 Application User (App)

All data consumers, who access the data through the provided API are considered to be application users in the context of Smart citizen assistant. This means, typically these are the Apps that are targeted to end-users, but any desktop or browser applications could also be seen the users. Based on the privacy requirements, there are two types of application users:

- Data-owners as users: Whenever data is consumed by the data-owners themselves, it is important to make sure that the credentials of the end-user are checked before the data is delivered. This could be in the form of login in the application or some other kind of token and key management technology. For example, whenever a smart meter user wishes to access the data collected from her smart meter, and then a login mechanism should control and verify that the user is allowed to access these data sets.
- Data consolidators as users: Whenever data is requested by users, who do not own the data, the platform must make sure that the data delivered in this context is anonymized and it is not possible to trace the owner of the data. For example, whenever the statistics department wishes to analyses the data of electricity usage in a city, and wants to access the smart meter data, then the data delivered in such a case should not contain information about the individuals.

2.2 Goals of Smart Citizen Assistant

The smart citizen assistant is comparable to an API manager, which stores, secures, scales and controls access to APIs to provide a resilient and flexible API runtime infrastructure. It also provides the analytics and reporting which may be needed by data publishers to act on trends that may potentially impact the way, in which data is published. For example, by understanding how developers are using APIs, the publishers can act on those insights to deliver increased value and API performance. API management software tools typically provide the following functions:

- Automate and control connections between an API and the applications that use it.
- Ensure consistency between multiple API implementations and versions.
- Monitor traffic from individual apps.
- Provide caching mechanisms to improve application performance.

- Protect the API from misuse by wrapping it in security procedures and policies.

2.2.1 Database of all Available APIs

In order to create a community of developers wishing to exploit the data available in a city, the smart citizen assistant acts as an API manager. It provides a platform for developer interactions, where developers can browse through all APIs available in the API database. The browsing experience could be improved by providing a graphical experience similar to Android Marketplace.

The database allows developers to browse and search APIs by provider, tags, or name and browse API documentation, download tutorials for easy consumption. Depending on the API and the type of data, it may also be possible to try APIs directly on the front-end. The interaction with the other developers is facilitated by allowing commenting and rating APIs.

Apart from that, the API manager also takes care of registration to developer community to subscribe to APIs. The subscription could be on per-application basis, or at a per-developer level. The API manager provides role based access to API database and has clear policies about how public and private APIs are managed.

Data publishers can interact with the API manager to view consumers' API analytics and draw insights on how the API can be improved. Metrics can be viewed on a user, API, or application basis.

2.2.2 Publishing and Governance

The first step in driving API usage is related to designing an API and publishing it for consumption in a publicly accessible platform. Together with the API and the service implementation of the API, the publisher also attaches documentation (files, external URLs), and defines the security and privacy policy associated with the API.

By managing API versions and deployment status, lifecycle of the API (publish, deprecate, and retire) can be governed by the publisher. The publisher also has the possibility to change the privacy requirements, after an API is published which may result in revoking access tokens from developers, who may already be using the API.

Upon successful deployment, the publisher is able to track consumer analytics per API, per API version, and per consumer. This gives some insights into the reality of how an API is used and may support decision processes in adapting or evolving the API. Such functionality may be provided in the form of alerting or real-time dashboards for publishers.

Developers who wish to use the API manager and develop new applications on top of this infrastructure must sign-up for API consumption. All developers in the community can communicate with each other through shared forums and comments they post for rating the available APIs.

2.2.3 Application Runtime Support

At runtime, whenever applications demand for data through the smart citizen assistant, the API manager is responsible for end-user authentication and developer/app authentication. The

service may only be invoked, if all the security policies are satisfied and the required credentials are provided. Therefore, the smart citizen assistant takes care of developer and end-user authentication, access control to service invocation and enforcement of API-specific privacy settings whenever required. For example, when anonymization must be performed before the data can be delivered to the developer, such tasks can be performed by the smart citizen assistant or the service implementation of the API itself.

2.3 Architectural Constraints

The architecture is to be defined such that the tools and technologies developed in this project are viable and continue to serve to the community of smart citizens even after the duration of the current funding period of the project (2016). This means, the smart citizen must build upon existing data sources and aim to integrate itself in the IT infrastructure, which will exist even after the project transform+ ends. Some of these infrastructure elements are described here as the constraints, which must be fulfilled for a long term success of the project.

As shown in the requirements description, there are multiple sources of data that must be integrated and considered during the implementation of the smart citizen assistant. For example, this could mean the integration of existing end-user databases (from Wiener Netze), existing smart meter database (Wiener Netze) or infrastructure in Aspern (from the research agency ASCR).

The existing infrastructure elements represent an opportunity for Transform+ and the smart citizen assistant to exist beyond 2016. Organizational, financial and legal issues around the use of these infrastructures need be clarified. These issues are clearly out of scope in this document, as we focus on the technical solution architecture. The open issues are listed in the end of this document, along with a plan for the resolution of the issues.

2.4 Security and Privacy Obligations

As the smart citizen assistant "processes" different types of data in the context of the city, including personal data, strict data protection policies are required to protect personal data and these must be implemented. This includes the commitment to ensure that data is collected and used fairly and lawfully – among others through the following means:

- Establishing appropriate retention periods for personal data: Personal data processed for any purpose shall not be kept for longer than is necessary for that purpose. This means, personal data is retained by the system only when the user is actively using the application, i.e., the user is logged in. After a proper log-out from the system, the data is also removed.
- Providing adequate security measures to protect personal data: Whenever personal data is retained and saved in the system, these are encrypted to ensure that no third party software or persons may access these. Furthermore, appropriate technical and organizational measures shall be taken against unauthorized and unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.

In the case of private or personal data, the publisher of the data must ensure that the data owners have provided their consent to process their data. This means, it is the responsibility of the data publisher to respect the privacy of the data subjects. The smart citizen assistant cannot detect whether the processed data is private or public. This is mainly because, the software component is agnostic about the content of the processed data, and rather it is only a technical infrastructure for doing so.

The end-user of the applications must declare that she agrees to the use of data relating to her in a given case, after having been informed about the prevalent circumstances, and must do so without constraint. The end-users have the right to know what information is being held about them, what their information is being used for and why, where the information came from and who has accessed it, request access to their information, take action to block, rectify, erase or destroy inaccurate data.

2.5 Voluntary Provision of Data

It is important to note, that the publishers of the data can only make the data available through the smart citizen assistant, which has been voluntarily made available by the end-user- Similarly, the developer can only access data that has been voluntarily made available by the data publishers. In the future, there could be a provision to actively request data by the developers, but this is not envisioned at the moment. As shown in the figure below, the sensors and meters in the house send the data to the utility providers. They can then make the data available through the smart citizen assistant. However, at each level of data transfer, legal regulations must be followed. Smart citizen assistant alone does not have any provision to check the legal issues around data provision and ownership.

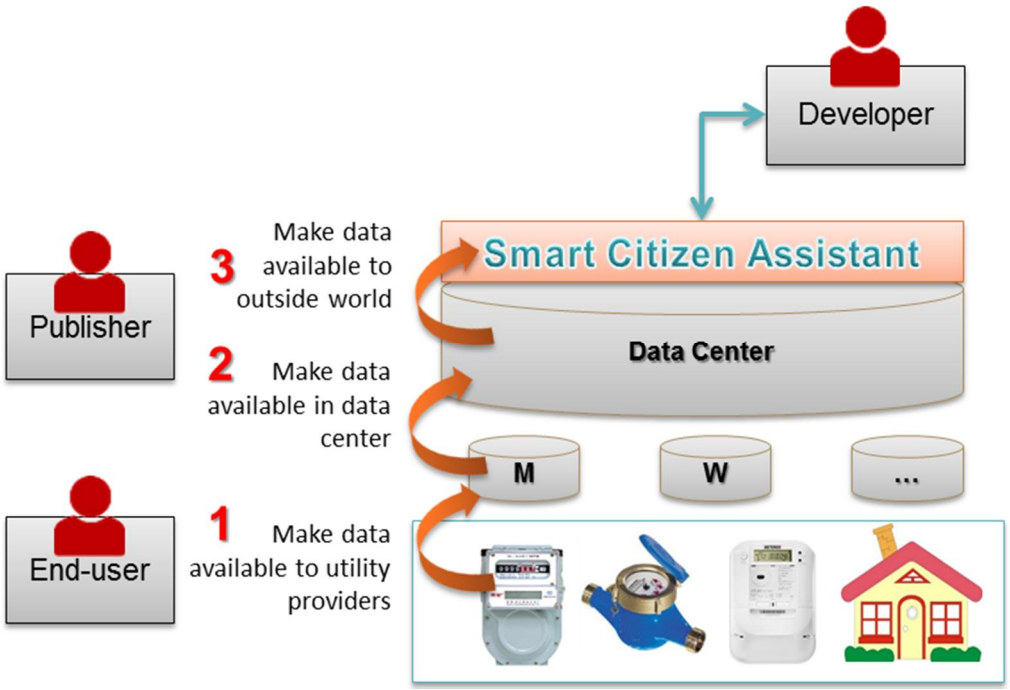

Figure 3 Availability of the data through the smart citizen assistant is based on voluntary provision

3 Architecture Definition

When defining the architecture of a software system, it is important to consider different aspects of the system, including the main functional elements, interactions between these elements and with the outside world, the kinds of information will be managed, stored, and presented, physical hardware and software elements will be required to support these functional and information elements. In this chapter, we describe all these important aspects from two different viewpoints (functional, implementation).

3.1 Functional Viewpoint

The functional view of the smart citizen assistant defines the architectural elements that deliver the system's functionality. It documents the system's functional structure-including the key functional elements, their responsibilities, the interfaces they expose, and the interactions between them. Taken together, this demonstrates how the system will perform the functions required of it. In the case of smart citizen assistant, the offered functionality can be divided into five major blocks of functionality (see Figure 4).

3.1.1 API Definition and Publishing

A set of functions are provided by the smart citizen assistant to enable API definition and publishing. This includes functionality to store different types of internal data:

- API Management is about providing an internal data storage and management capabilities to store, administer, search and explore the API that are currently available.
- Publisher Management is about administering the users, who can register themselves as publishers and add new APIs (including data access services) to the platform.
- Developer Management is about internal records of developers who are registered as API users. The system provides mechanisms for developers to subscribe to APIs and be informed about the changes.
- User Management is about taking care of the end-user credentials, whenever the API or APPs attempt to access personal data. The smart citizen assistant makes sure, that the necessary policies are enforced.
- Data Access Services are the implementations of the published APIs that can be accessed in through the smart citizen assistant in a controlled manner.

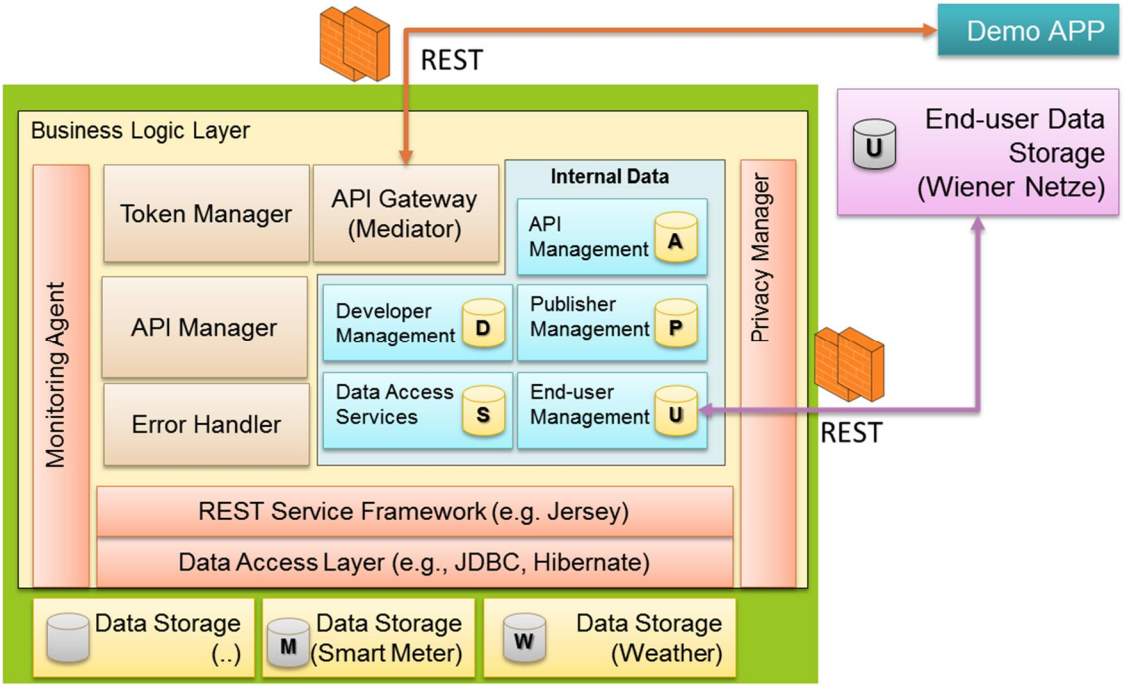

Figure 4 Functional viewpoint on the smart citizen assistant architecture.

3.1.2 Runtime Services

At runtime of applications and APPs, the smart citizen assistant provides services that are crucial for accessing the data required by the APPs—thereby making sure that the privacy requirements are met. In particular, the following functionality is provided:

- Invocation Service is the gateway from the app to the smart citizen assistant. This service is responsible for passing the requests from the client s to the corresponding services and calling other services in the background to ensure security and privacy.
- Authorization Service takes care of validating the credentials of publishers, developers and end-users whenever they interact with the smart citizen assistant.
- Token Manager takes care of persistent authorized status of all users based on keys and tokens that are generated and remain valid for a certain period of time.
- Privacy Manager is responsible for applying the correct privacy policies on the different types of data (personal, general) and for different types of data access stakeholders (through data-owners and anonymized access).

3.1.3 Monitoring and Analytics

All the interactions between the different stakeholders (users, application) through the smart citizen assistant are monitored and tracked by a monitoring agent, which can store the data for different analytic functions.

- Monitoring Agent is the component that tracks and records the data about all transactions.
- Error Handler is evoked, whenever the system detects that some kind of problems occurred at runtime.

- An Analytics component takes care analyzing the monitored data and creating different reports for different stakeholders (developers, publishers etc).
- The Dashboard displays different statistics and reports generated by the analytics component and makes these available through a browser.

3.1.4 Data Storage

The actual data (e.g., Smart meter data, weather data) is stored in different databases, which may or may not be located in the same physical location as the smart citizen assistant. The goal is to allow third parties to be part of the integral system by integrating their databases through the smart citizen assistant.

3.1.5 External Tools

The applications that build upon the smart citizen assistant and different community forums are seen as external tools, but are still considered to be part of the smart citizen assistant integral system.

3.2 Implementation Viewpoint

From the technical perspective, the architecture is essentially a layered architecture where the upper components depend (or use) only on components directly beneath them. The big exception in our architecture is the monitoring component which can be used by any component. In the rest of this subsection we detail the individual components. The interface descriptions of the different components are often given in the form "*message*(*parameters*)"; where *message* denotes the name of the message or method and *parameters* is a list of parameters needed for one particular method call.

An overview of the components is depicted in Figure 5. In this section we describe the individual components and their functionality from the implementation perspective.

3.2.1 Components

- Web Presentation : The Web Presentation component provides a web interface to the API store. This web interface especially allows different kinds of user to register and manage their data. This component will use the APIs exposed by the REST framework below, specifically those which are exposed by the smart citizen assistant.
- REST Framework: The REST Framework manages the communication and translates between REST communication and mediator component. The components interfaces are used by consumers and the Web Presentation component directly.
- Mediator: The mediator routes requests from the REST framework to API calls. The mediator manages and executes these routes.
  - `CallService(serviceid,methodname,serviceparameters,apiparameters)`
  - `NewRoute(serviceid,url pattern,methodname,apiparameters)`
  - Calls to the different management components

The *serviceid* is an ID corresponding to a specific service implementation. The *serviceparameters* is a list of parameters interpreted by the service implementation itself. The *apiparameters* contain different parameters needed for the API management such as username, password, different kinds of tokens and keys.
The Mediator and the REST Framework components are logically different. In practice they might be tightly coupled.

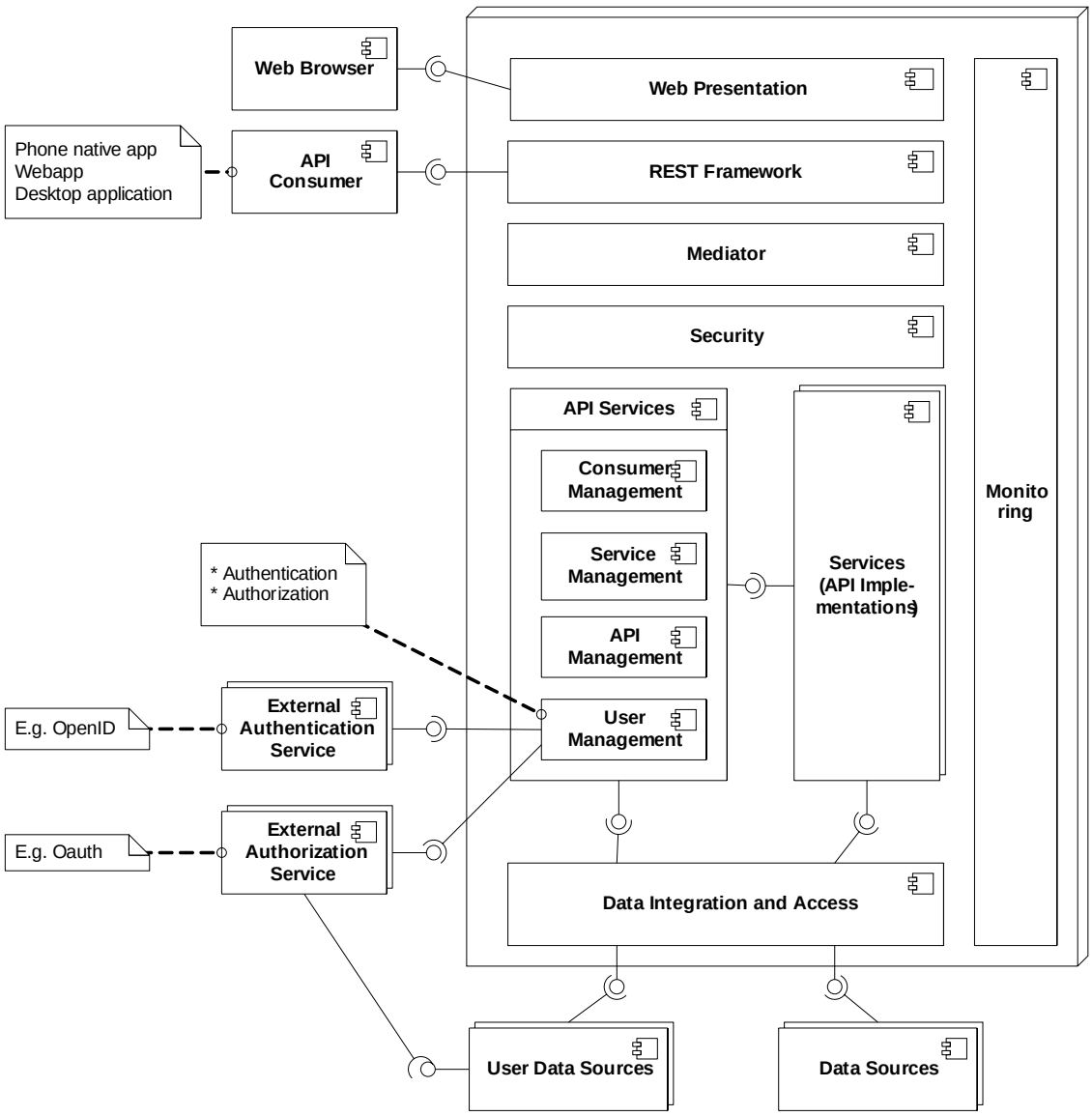

Figure 5 Component diagram of the smart citizen assistant and their interfaces

- Security: The Security component enforces the security policy. In particular it provides authentication and authorization services, and ensures privacy.
  - `Boolean Authenticate(username, password)`
  - `Boolean Authorize(userid, consumerid)`
  - `IsConsumerApproved(consumerid)`
  - `IsServiceApproved(serviceid)`
  - `MayUserUseConsumer(userid, consumerid)`

  - `MayConsumerUseService(consumerid, serviceid)`
  - `Call(service,serviceparameters,apiparameters)`
- API Management: The API Management component delivers services to manage APIs. An API is an abstract description of a service interface. These descriptions are mainly useful for consumer developers when looking for APIs they can use and for publishers.
  - `ListAPIs()`
  - `FindAPI(apiid)`
  - `SearchAPI(search string)`
  - `SubmitNewAPI(name,version,spec)`
- Service Management: The service management component allows a publisher to list find and submit new services. A service implements one ore more APIs.
  - `ListServices(apiid)`
  - `FindService(serviceid)`
  - `SubmitNewService(list of apiids, implementation)`
  - `GetServiceUsageStatistics(serviceid)`
- Consumer Management: The consumer management component allows a developer to list, find, and submit consumers for approval. The developer can also get consumer usage statistics.
  - `ListConsumers(developerid)`
  - `FindConsumer(consumerid)`
  - `SubmitNewConsumer()`
  - `GetConsumerUsageStatistics(consumerid)`
- User Management: The user management component allows a user to register, authenticate, and change personal information.
  - `Boolean Authenticate(username, password)`
  - `Boolean Authorize(userid, consumerid)`
  - `CreateUser(username, email, password, role)`
  - `ChangePassword(userid,oldpassword,newpassword)` : Change personal information such as passwords
- Monitoring: The monitoring component logs and different kinds of actions, e.g., a newly registering user, a user accessing a service.
  - `Log(Level,userid,role,Message)`
  - `GetConsumerUsageStatistics(consumerid)`
  - `GetServiceUsageStatistics(serviceid)`
- Data Integration and Access: The Data Integration and Access component provides an integration layer over the specific data sources. This component allows transparent access to metadata DBs used by the SCA itself as well as service specific DBs: This component gives a seamless view over several databases allowing the API services high-level data access without considering integration of physical data sources. The data model of the API services is depicted in Figure 6.
- Data Sources: The Data Sources components are different kinds of persistence services. Generally we can distinguish between user dependent data sources and user independent data sources. For user dependent data sources, the publisher has to implement some kind of authentication and/or authorization service (e.g., based on OAuth) which allows an API and thus a user to access her and only her data.

3.2.2 Data Model

An overview of the data model is given in Figure 6. It describes the main concepts used by the SCA and their corresponding relations.

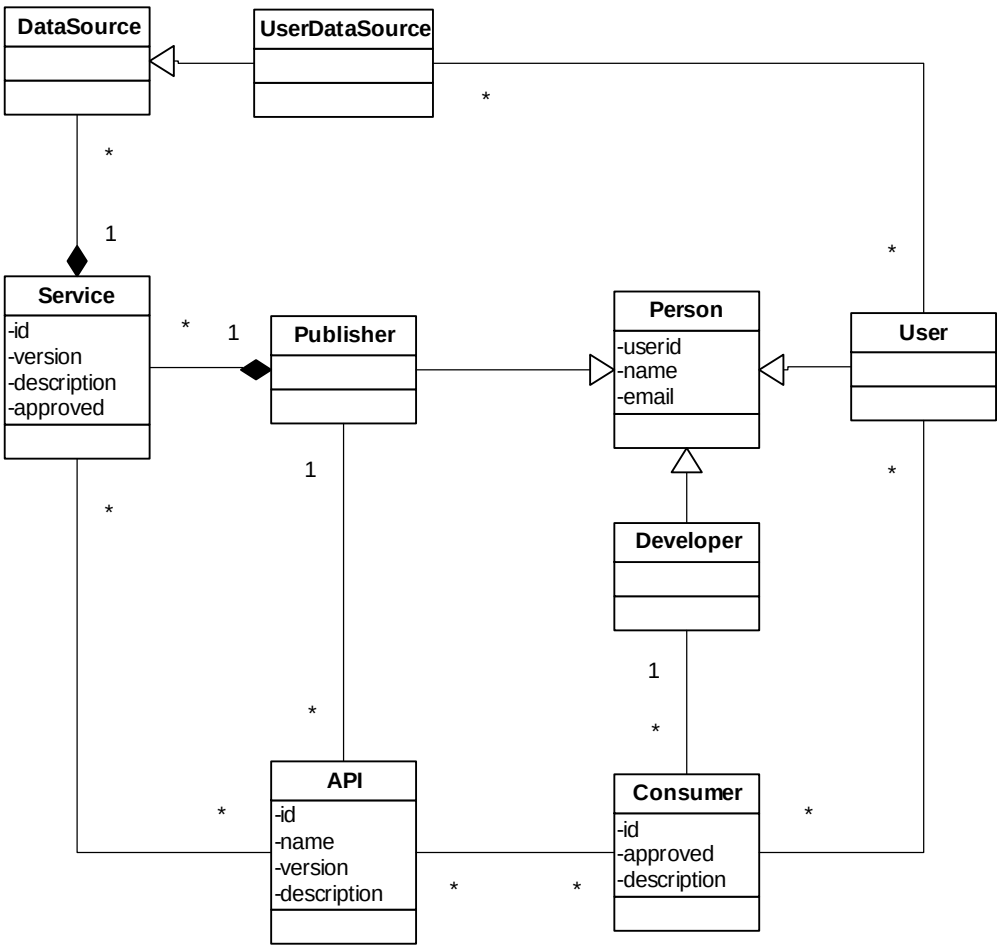

Figure 6 Data model for storing the APIs – an integrated view used by the smart citizen assistant.

4 Interactions and Workflows

A workflow defines a set of correlated activities each of which representing a (composite) component that takes input, transforms it, and sends the result via its output ports to subsequent activities. The interactions may be triggered implicitly by (human or machine initiated events) or by explicit transfer of control. As described in the previous chapters, different human and machine stakeholders interact with the smart citizen assistant. In this chapter, we describe on the level of technical components, how these interactions look like and how these can be implemented.

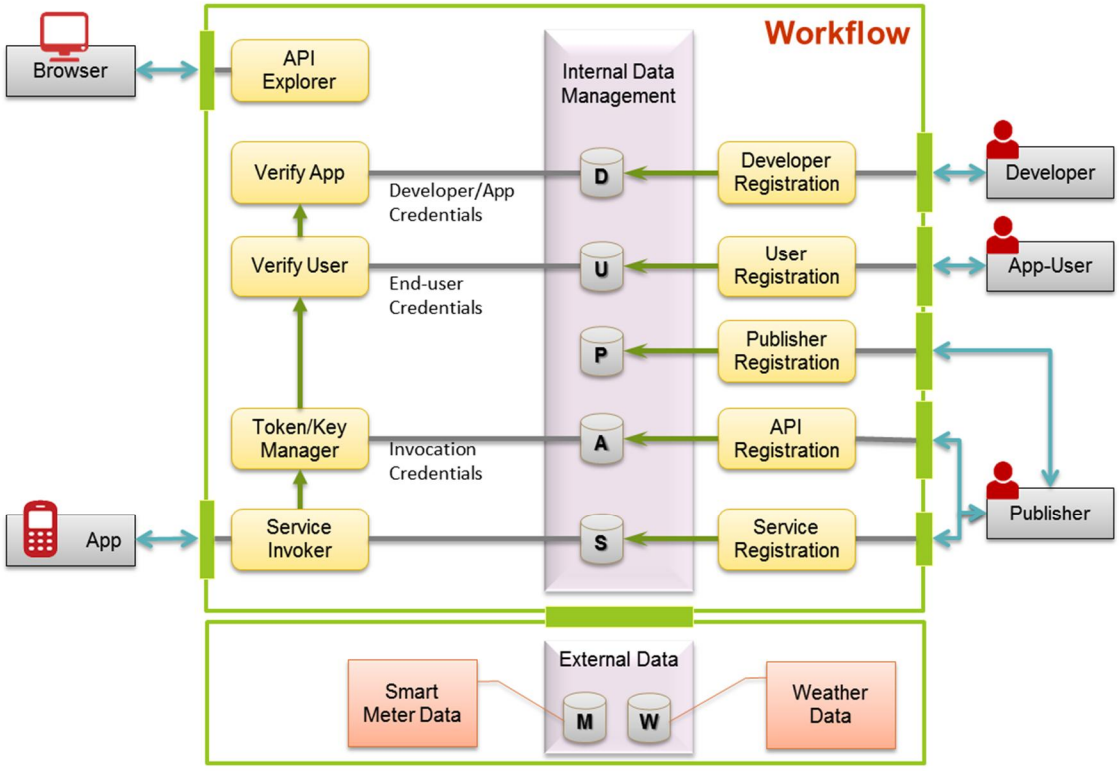

Figure 7 An overview of involved stakeholders and their interactions with the smart citizen assistant.

As depicted in Figure 7, the workflows within the system are pretty complex and involve several internal data management components:

- [D] Developer Data Management: Data about the registered developers, who are entitled to use the API and services to access the data.
- [U] End-user Data Management: Data about the end-users who use the APPs and who have given their consent to use or process their personal data.

- [P] Publisher Data Management: Data about the publishers of the data, including their declaration of why they are allowed to publish the data.
- [S] Service Data Management: Data about the available services, their deployment locations etc.
- [M] Smart Meter Data Management: Data collected from smart meters, owned by Wiener Netze.
- [W] Weather Data Management: Data from an external weather data provider.

4.1 General Codes

For REST the usual HTTP status codes are used to indicate errors on the server or client side. The following table lists the used HTTP status codes and their explanation in the context of this architecture.

HTTP Status Code	Explanation
200	OK – successful request
201	Created – resource (e.g., API) has been successfully created
400	Bad Request – The response body will contain a structured explanation to allow clients to resend their request. Often one of the parameters is invalid
401	Unauthorized – the user has to authenticate first
403	Forbidden – the request is forbidden, regardless of authentication.
404	Not found – the resource could not be found. This especially will happen for resources like specific APIs if the apiid in the request is unknown to the system
405	Method not allowed – the request method (GET, POST, DELETE, PUT) is not allowed for this resource
500	Internal server error

Next we describe all the different kinds of interactions grouped by the three different roles publisher, developer and end-user.

4.2 Publisher Interactions

A Publisher provides some kind of data or service implementation according to an API specification. This section describes the three different interactions possible for a publisher, which are publisher registration, API registration, and providing a service implementation.

4.2.1 Publisher Registration

Before logging in the first time a publisher, like any other kind of user has to register with a username, email address, and a password.

If the publisher registration is successful the REST service will return a new publisher ID. This publisher ID will not be active after the POST request. The publisher will receive an email to the given email address with a verification code. Only after verifying the email address a publisher ID will be active.

If the publisher registration is not successful (e.g., incorrectly formatted email address, or weak password) a structured response will be returned by the service.

REST Specification:

HTTP Method	POST
URL	/publishers
Access policy (role)	-
Parameters/Payload	Username – String Email – String (email address) Password – String (strong password)
Result (Success)	Publisherid
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.2.2 API Registration

Before a service implementation (or API implementation) can be uploaded, the corresponding API description has to be uploaded. We call this function API registration. This service request must have the following parameters: a unique name for the API, the ID of the requesting publisher, and a structured specification of the API.

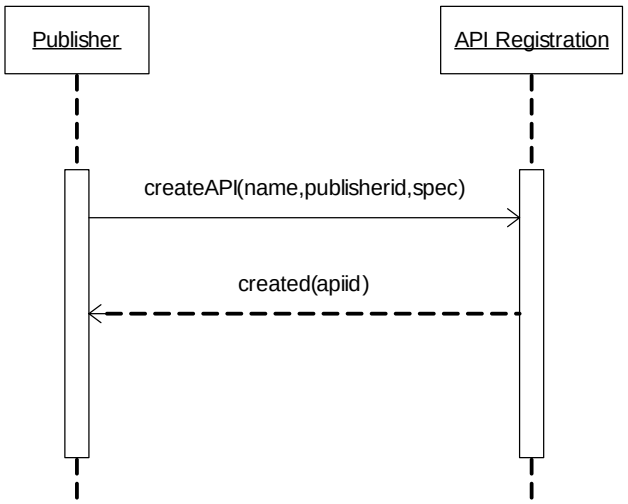

Figure 8 Sequence diagram for developer registration

If the request is successful a new API ID will be generated and returned in the response.

If the request is not successful (e.g., already used API name, invalid publisher ID, invalid API specification) a structured error response will be returned.

REST Specification:

HTTP Method	POST
URL	/apis
Access policy (role)	Publisher
Parameters/Payload	Authentication token apiname – String publisherid – PublisherId apispec – APISpec
Result (Success)	Apiid
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.2.3 Provide service implementation

After one or more API specifications are registered, a publisher can upload/provide a service implementation.

The following parameters are needed: a unique name of the service, the service implementation in binary form, a list of one or more API IDs. The service implementation must implement all the given APIs.

If successful the response will contain a new service ID.

If unsuccessful (e.g., name not unique, invalid implementation, invalid API ID), a structured error response will be returned.

REST Specification:

HTTP Method	POST
URL	/services
Access policy (role)	publisher
Parameters/Payload	Authentication token servicename serviceimplementation list of apiids
Result (Success)	Serviceid
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.3 Developer Interactions

A developer uses service implementations through one or more APIs to develop some kind of (data/service) consumer.

4.3.1 Developer Registration

Before logging in the first time a developer, like any other kind of user has to register with a username, email address, and a password.

If the developer registration is successful the REST service will return a new developer ID. This developer ID will not be active after the POST request. The developer will receive an email to the given email address with a verification code. Only after verifying the email address a developer ID will be active.

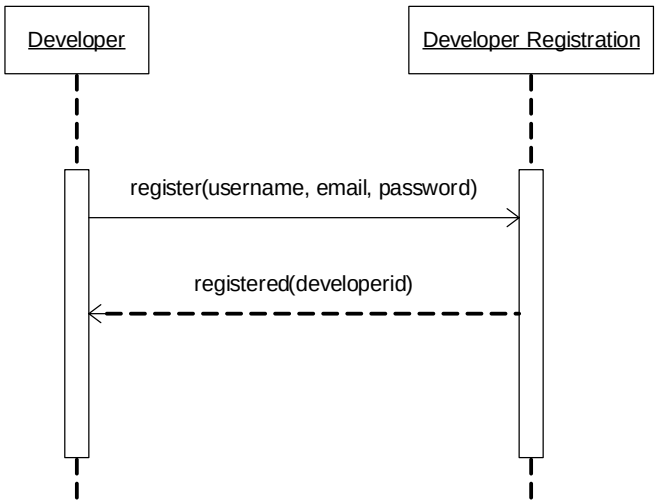

Figure 9 Sequence diagram for developer registration

If the developer registration is not successful (e.g., incorrectly formatted email address, or weak password) a structured response will be returned by the service.

REST Specification:

HTTP Method	POST
URL	/developers
Access policy (role)	-
Parameters/Payload	Username – String Email – String (email addresss) Password – String (strong password)
Result (Success)	Developerid – String
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.3.2 Consumer Registration

When developing a consumer application a developer has to register the consumer to obtain a consumer ID (also known as API key from other web API providers). With this consumer ID the consumer will then be later able to use the different APIs.

The needed parameters are the following: a unique consumer name/version combination and a list of API IDs the consumer will be accessing.

If the consumer registration is not successful (e.g., invalid API ID) a structured response will be returned by the service.

REST Specification:

HTTP Method	POST
URL	/consumers
Access policy (role)	Developer
Parameters/Payload	Authentication token name - String version – String list of apiids
Result (Success)	Consumerid – String
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.3.3 Browse APIs

This REST interface can be used to browse through all the APIs provided by the publishers. The result is a list of APIs with basic information and an API ID which can afterwards be used to get more details.

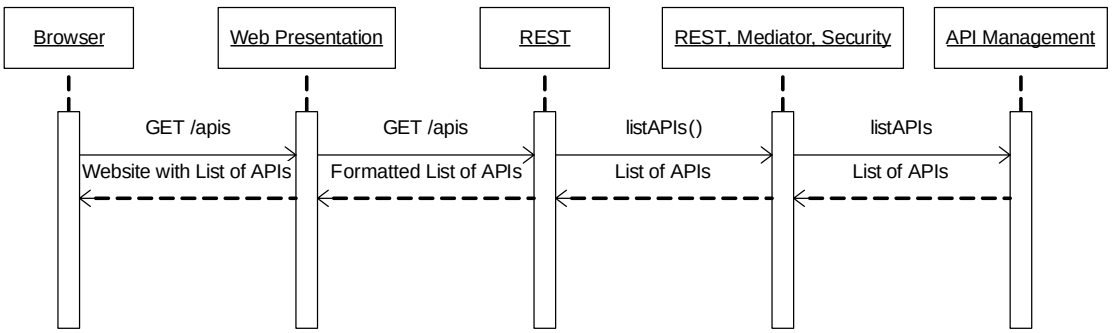

Figure 10 Sequence diagram for API exploration

This API call does not need any parameters (besides the authentication token).

The response will contain a list of API IDs together with a corresponding name and URI. The URI in the response can then be used by the following interactions for getting details of a specific API.

REST Specification:

HTTP Method	GET
URL	/apis
Access policy (role)	Developer
Parameters/Payload	Authentication token
Result (Success)	List of apiid - String apiname - String apiurl – URL
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.3.4 Search for API

A developer can also search for an API with some search string. The result is a list of APIs with basic information and an API ID which can afterwards be used to get more details.

The parameter, transmitted in the URI is a simple search string. The service will search for the string in the API specifications.

The results format is the same as for the “Brows APIs” interaction. Only for this interaction the service will only return APIs which contain the search string.

REST Specification:

HTTP Method	GET
URL	/apis?search={searchstring}
Access policy (role)	developer
Parameters/Payload	Authentication token
Result (Success)	List of apiid - String apiname - String apiurl – URL
Result (Error)	

4.3.5 Get Info on specific API

After finding an interesting API with one of the API calls above, a developer can get details of a concrete API with this REST call.

The only parameter (in the URI) is a concrete API ID.

If the API with the given API ID exists and is accessible for the currently logged in user, the service will respond with an API specification.

If unsuccessful (e.g., incorrect API ID), the service will answer with a structured error response indicating the error.

REST Specification:

HTTP Method	GET
URL	/apis/{apiid}
Access policy (role)	developer
Parameters/Payload	Authentication token
Result (Success)	APIspec
Result (Error)	HTTP 404 if the API cannot be found HTTP 407 if

4.4 End-user Interactions

End-users use the applications developed by a developer to access general or personal data.

4.4.1 Registration

Before logging in the first time a user, like any other kind of user has to register with a username, email address, and a password. If the user registration is successful the REST service will return a new user ID. This user ID will not be active after the POST request. The user will receive an email to the given email address with a verification code. Only after verifying the email address a user ID will be active.

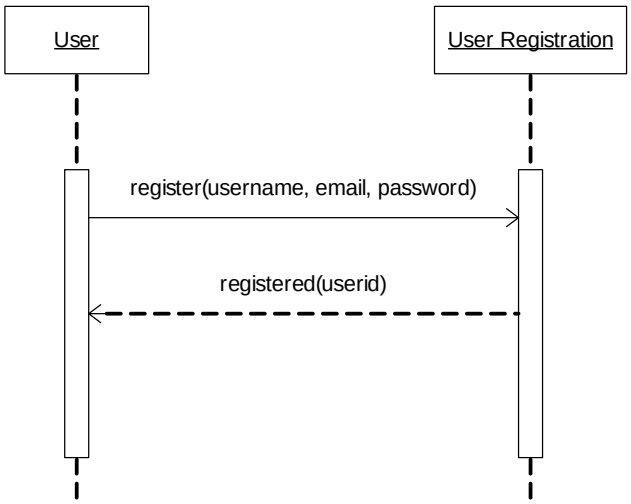

Figure 11 Sequence diagram for end-user registration

If the user registration is not successful (e.g., incorrectly formatted email address, or weak password) a structured response will be returned by the service.

REST Specification:

HTTP Method	POST
URL	/users
Access policy (role)	-
Parameters/Payload	Username – String Email – String (email addresss) Password – String (strong password)
Result (Success)	Userid
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.4.2 Use Service

This interaction covers the use case of an end user accessing some API/service implementation. The needed parameters are specific to the implemented API. The response is service and API specific.

REST Specification:

HTTP Method	GET/POST/PUT/DELETE
URL	/services/{serviceid}/{servicespecific}
Access policy (role)	User
Parameters/Payload	Authentication token Api specific
Result (Success)	Api specific
Result (Error)	HTTP 400 + structured reason if one of the parameters is invalid

4.5 End-user Authentication and Authorization

We use OAuth for Authentication and Authorization. Figure 12 shows an example of how a consumer (in this case a smart phone app) can get a authentication token from the server.

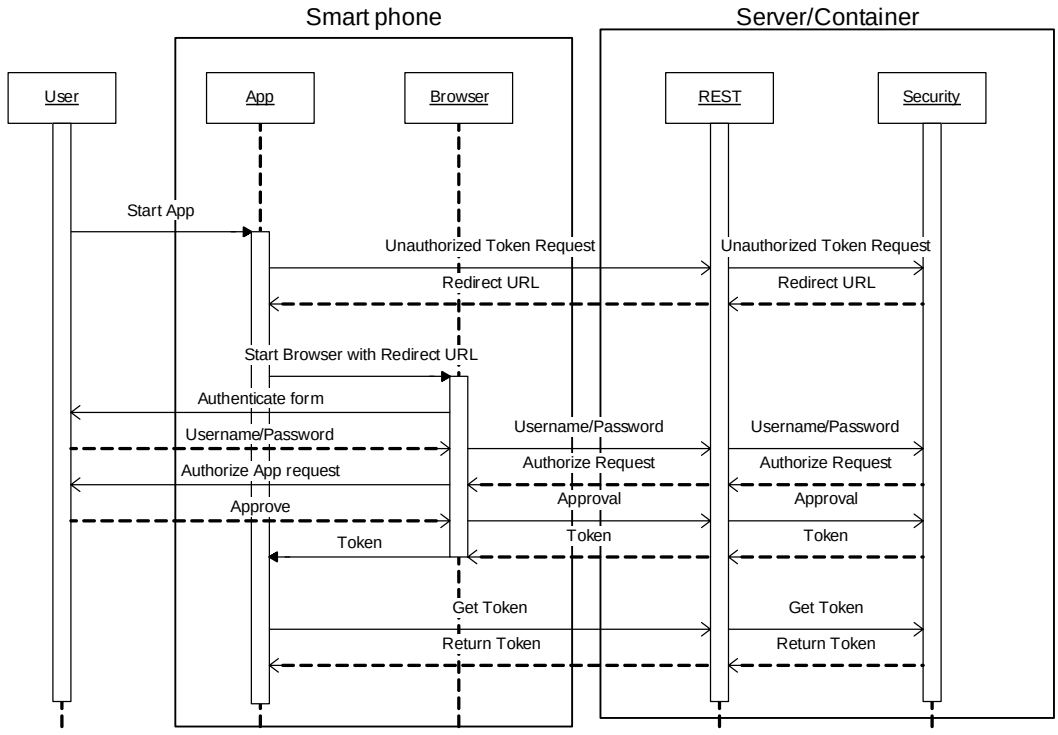

Figure 12 Example of Authentication and Authorization and a single API call from a smart phone app

When the APP is started it sends an *Unauthorized Token Request* to the server. The server then sends a URL to the consumer. The consumer has to open a web browser with this URL which allows the user to enter login details and authorize the service. Eventually when authentication and authorization are successful the consumer will retrieve an authentication token which needs to be submitted with every future request.

4.6 Service Invocation

Figure 13 shows how a service call runs through all the layers down to the concrete service implementation. We assume a service implementation *@123* implementing a weather lookup. The consumer (which authenticated earlier and authorized the service) connects to the REST framework and sends a REST request. The REST request is, as always, encoded as a URL asking for the weather in Vienna in Austria. The consumer knows from previous calls to the REST interface that the needed service has the ID *123* and the wanted method is called *weather*. The consumer sends the required information such as userid, consumerid along in the request.

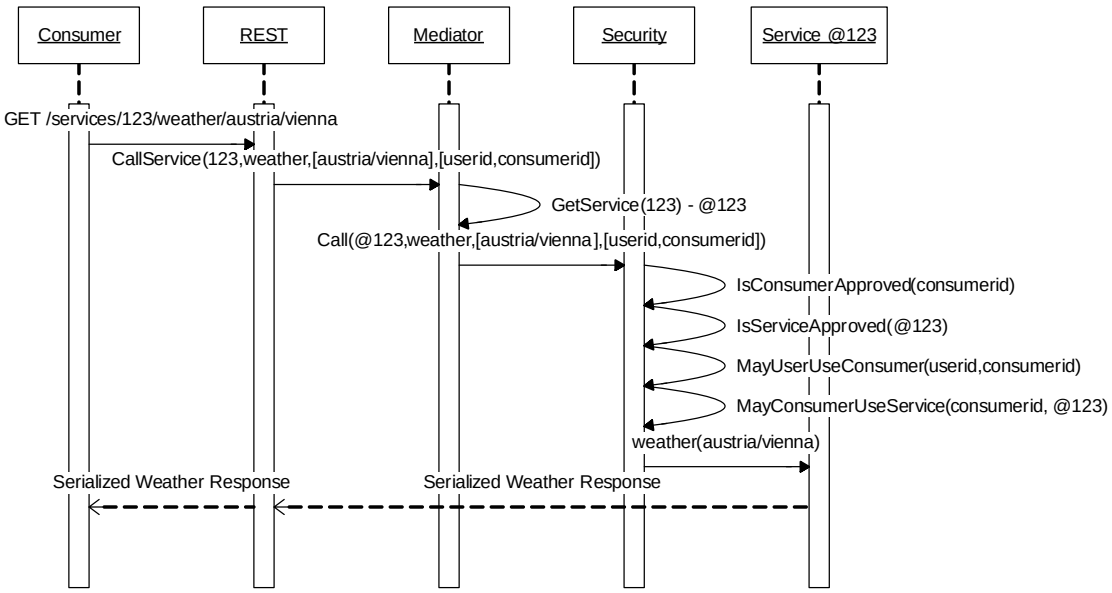

Figure 13 API call from consumer illustrating the roles of REST framework, Mediator, Security and a concrete Service implementation

The REST framework parses the request and calls the Mediator to further process the request. The Mediator looks up the service implementation for the service ID 123 (maybe also through the Data Integration and Access component; not shown here for brevity).

After finding the service implementation @123 the mediator calls the Security component to further process the request. The Security component then checks conformance to the security policy of the request. Depending on the consumer, service, and user, different kinds of checks are executed.

If and after all the checks are passed the Security component eventually calls the service implementation *@123* with the specified method *weather* and the handed over parameters *austria/weather* (see Figure 13). See

The service @123 processes the requests and, for our example, looks up the weather of Vienna (maybe through the Data Integration and Access component) and returns a formatted response to the REST framework.

4.7 Service invocation for security policy violation

The same process is executed until the Security component

If one of the security checks fails, e.g., the consumer is not approved, the Security component reports a security incident to the monitoring component and returns a security error response to the REST framework (see Figure 14).

The REST framework sends the corresponding error response to the consumer, a HTTP 407 in this case, together with a short error description which might include next steps to resolve the problem.

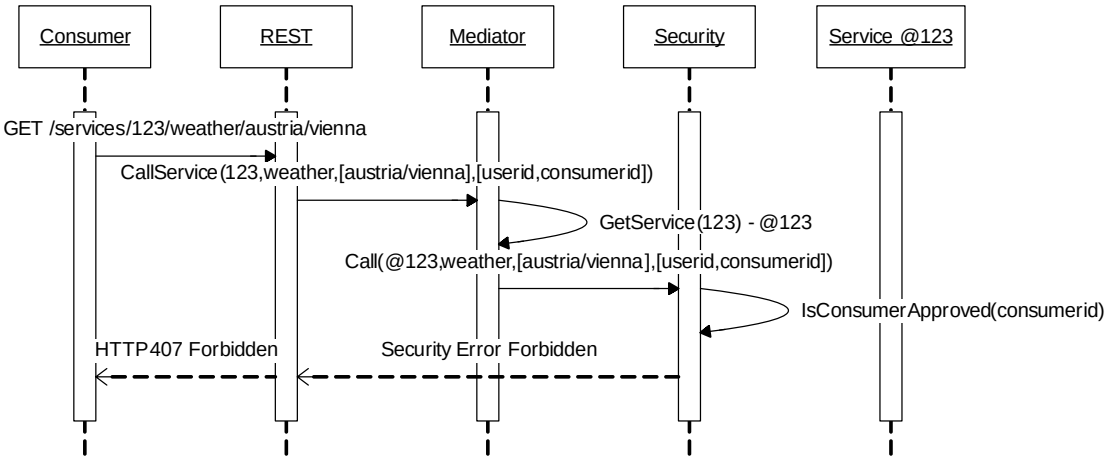

Figure 14 API call from consumer illustrating the roles of the different components in case of a security policy violation

5 Deployment Scenarios

Deployment of the different components and the integration of different data sources for the operation of the smart citizen assistant is a bigger problem from an organizational perspective rather than from a technical perspective. This chapter describes from the infrastructure perspective, where the different components of smart citizen assistant can be deployed.

As identified in this document, there are seven kinds of information sources (internal databases) that are required for the implementation. In order to ensure the viability of the smart citizen assistant after the runtime of the project Transform+ it is important to use the infrastructure that exists even after the project has ended. On the basis of these prerequisites and the possible sources of different data, four different deployment scenarios have been identified:

	Scenario 1 (best case)	Scenario 2 (can't access Wiener Netze Infrastructure)	Scenario 3 (can't access ASCR Infrastructure)	Scenario 4 (worst case)
D Developer DB	ASCR (seestadt Aspern)	ASCR (seestadt Aspern)	Transform Plus Developers	Transform Plus Developers
U End-User DB	Existing Wiener Netze User DB	Transform Plus users for real data from ASCR	Existing Wiener Netze User DB	Transform Plus users for Transform Plus data
A API DB	ASCR (seestadt Aspern)	ASCR (seestadt Aspern)	Transform Plus API Store	Transform Plus API Store
M Smart Meter Data DB	ASCR Teradata	ASCR Teradata	Transform Plus Meter Data Database	Transform Plus Meter Data Database
P Data Publisher DB	ASCR (seestadt Aspern)	ASCR (seestadt Aspern)	Transform Plus Publisher DB	Transform Plus API Store
W Weather Data DB	ASCR Teradata	ASCR Teradata	Transform Plus Weather DB	Transform Plus Weather DB
S Services DB	ASCR (seestadt Aspern)	ASCR (seestadt Aspern)	Transform Plus API Store	Transform Plus API Store

Figure 15 Summary of deployment scenarios for the different data management components

As the smart citizen assistant is a software component which needs resources for operation and maintenance, it is important to assign and allocate resources for continuous operation, maintenance, updates and upgrades, which are available even after the end of Transform+. Technically, this is possible by utilizing the infrastructure of Aspern Smart City Research (ASCR) and Wiener Netze. *In the worst case*, both these resources are not available to Transform+, which means the smart citizen assistant will not be available (due to lack of resources) after the end of the project. For this reason, the best scenario of deployment is where these existing infrastructure resources can be used and the results remain accessible.

5.1 Scenario 1: Smart citizen will exist after the end of Transform+

This is the planned scenario for deployment of the smart citizen assistant and it assumes that the legal, organizational and financial issues around using infrastructure from other projects can be regulated.

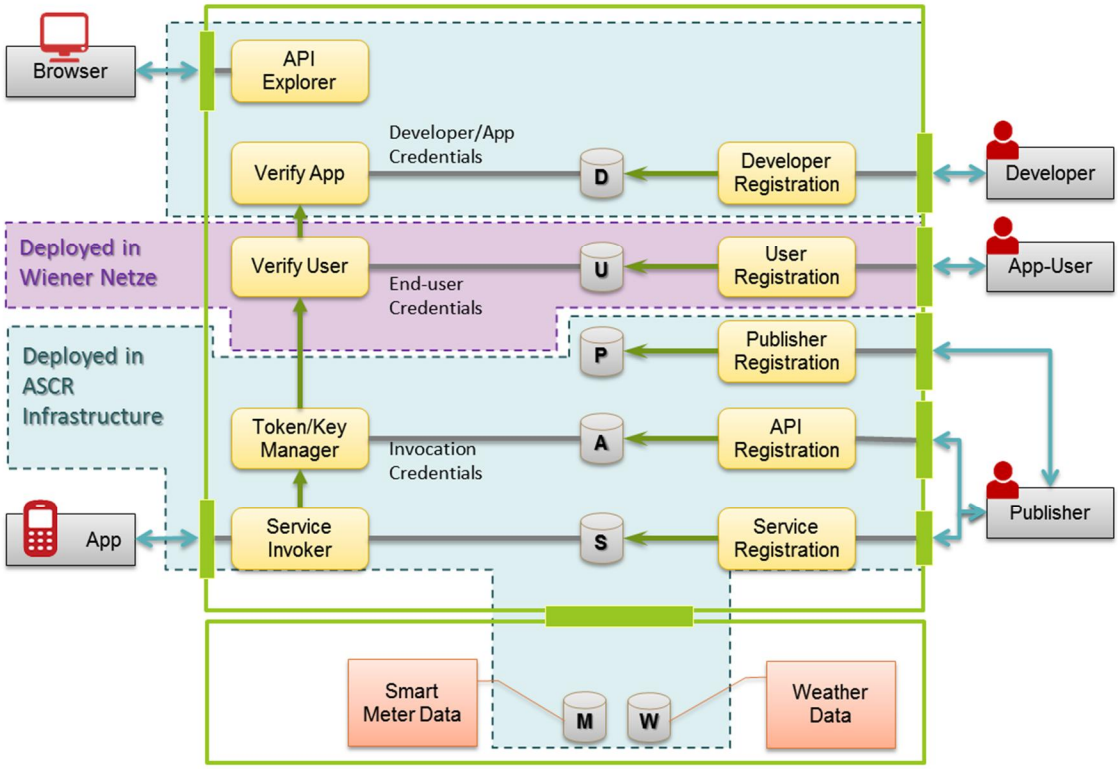

Figure 16 Ideal deployment scenario to ensure long-term success of the application.

In the ideal case, the smart citizen assistant will continue to exist even after the end of the project, as the deployment infrastructure for the SCA will be available beyond 2016.

The smart citizen assistant components and the data sources (weather, smart meter) are all deployed in the infrastructure of ASCR. In the pilot phase, the targeted end-users will be residents of Aspern Urban Lakeside, which nicely corresponds to the research goals and focus area of ASCR. However, the organizational issues around this must yet be discussed with the decision makers.

The details of the end-users are stored in the Wiener Netze data base. To protect the privacy of the end-users, the planned scenario considers using this information in the form of an authentication service, which could be deployed in the Wiener Netze Infrastructure. This means, the smart citizen assistant uses an existing service for authentication and relies on the service provider to clarify and deal with privacy issues of the end-users data.

5.2 Scenario 2: Wiener Netze Infrastructure cannot be accessed

This is a fall-back scenario, where the infrastructure of Wiener Netze cannot be used for the smart citizen assistant, with the assumption that the ASCR infrastructure can still be used.

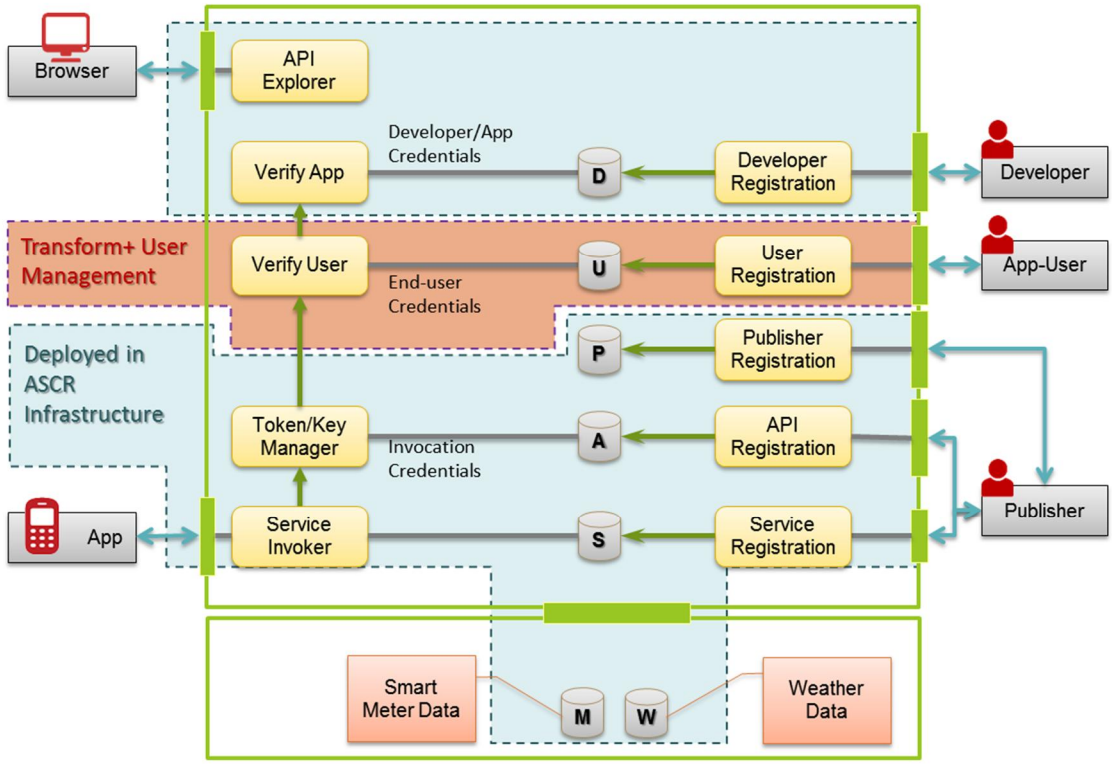

Figure 17 Deployment scenario without Wiener Netze Infrastructure

Similar to scenario 1, all the components for managing the API, developers, publishers and data access services will be deployed in the ASCR infrastructure.

As the end-users database from Wiener Netze cannot be accessed for authorization purposes, a "Transform+ database" of end-user data is required in this case. The Transform+ database contains generated end-user data for real smart meter readings saved in the ASCR data center.

This also means, that in this scenario, the smart citizen assistant cannot exist beyond 2016 to serve personal data, as the component required for end-user management will be exist after the end of the project Transform+.

5.3 Scenario 3: ASCR Infrastructure cannot be accessed

This is a fall-back scenario, where the infrastructure of ASCR cannot be used for the smart citizen assistant, with the assumption that the Wiener Netze infrastructure can still be used. This means, a dedicated server is required to execute the smart citizen assistant, and several data management components must be handled by this server (developers, publishers, APIs, services etc).

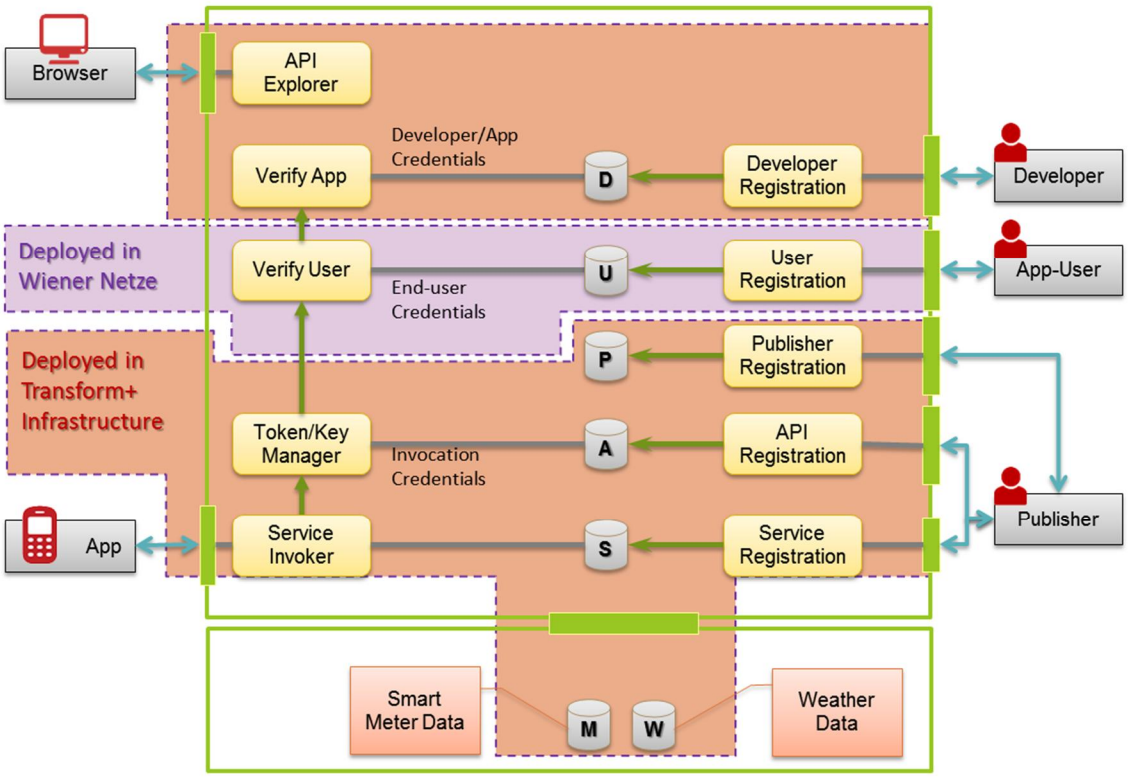

Figure 18 Deployment scenario without ASCR Infrastructure

The end-user authentication would still be done by Wiener Netze, but due to the lack of real meter readings (available in ASCR Databases), this scenario is not of much value to the end-user of the applications.

5.4 Scenario 4: Smart Citizen Assistant will only exist with Transform+

In the worst case, both ASCR and Wiener Netze Infrastructure cannot be used by the smart citizen assistant, meaning that it will have to run in its own server with its own end-user and data management. The demo apps can be used by real users but the data served by these APPS will not be real and so the total of the smart citizen assistant will be limited to demo data and demo users only.

Apart from that, even the demo app and the demo SCA infrastructure will not exist after 2016 mainly for the following reason: Smart citizen assistant is a software component which needs resources for operation and maintenance. In order to make this component available after the runtime of the project, resources must be assigned and allocated for continuous operation, maintenance, updates and upgrades.

In this case, the goal should be to find out organizations or companies interested in hosting the smart citizen assistant and making it available for public use. This is a challenging task, as the host of the smart citizen assistant must be able to develop a business case around this issue first. Only then, success of SCA is guaranteed.

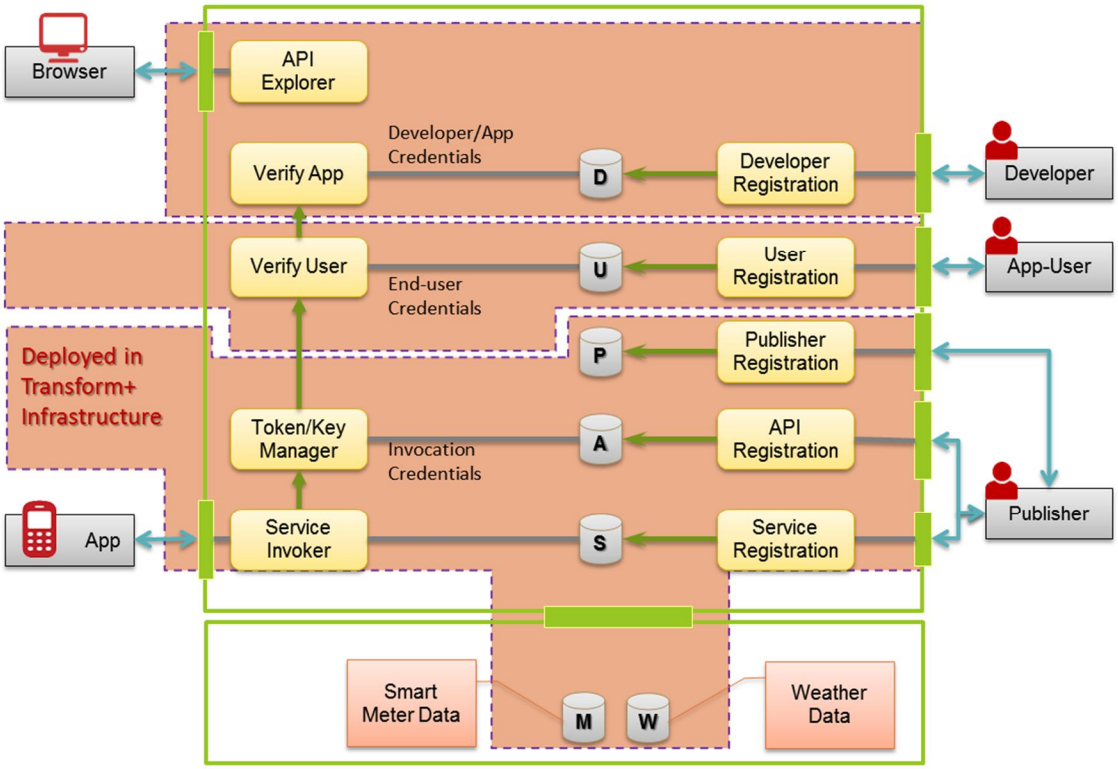

Figure 19 Deployment scenario without ASCR and Wiener Netze (Transform+ Setting)

5.5 Deployment Decisions

The decision about which scenario deployment scenario is realistically possible, will be taken by the decision makers in the involved organizations. The implementation of software components building up the smart citizen assistant is largely not affected by the decision (except for some additional connecters required in some cases). This means the overall technical solution is independent of the deployment scenarios.

Nevertheless, the decision about the availability and accessibility of the required sub-systems will be needed by the end of 2014. Discussions with the decision makers have started already at different levels.

6 Demo APIs and Applications

In order to demonstrate the value of the smart citizen assistant, two demonstrator APIs will be defined – and the corresponding services for providing the data will be implemented. A demonstrator APP will consume both the APIs and present the data from two different sources in an innovative manner to the end-user.

The two example APIs are Smart Meter Data API and Weather Data API. These are selected such that different aspects of the smart citizen assistant can be demonstrated. Smart meter data represent personal data and weather data represent general data, which are not directly protected by data privacy regulations. Apart from that, the smart meter data will be accessed through two variants of the API (end-user accesses personal information) and anonymized for statistics purposes (e.g., Wiener Netze accesses data for creating reports). Based on the two APIs, a smartphone APP will be designed to integrate the two different data sources.

6.1 Smart Meter Data API

The REST API is the underlying interface for connecting the mobile app to the SCA framework. In general the Smart Citizen Assistant platform has to provide two interfaces:

- REST API Master Data - API for login, user profile, master data, devices, etc.
- REST API metering values - API exclusively for querying measured values

The REST API metering values and master data are stateless and hold no session with the client. At each call, a HTTP Authorization header with valid credentials must be specified. In case of missing or invalid credentials a HTTP 401 status code is returned. Only a valid Authentication REST API can be used.

For the development of this web service according to the Representational State Transfer (REST) architectural style specification JSR-311 JAX-RS is used. As JAX-RS framework Jersey[3] is used. The HTTP body content is represented in JSON format to the Jackson[4] JSON parser which will convert Java objects to JSON and vice versa.

As already described in chapter 5, there are different ways to implement the necessary data sources which are specific for the demo app into the SCA platform. Depending on which one of the four scenarios will be chosen from the project team the infrastructure as a prerequisite for the demo APP will vary.

[3] https://jersey.java.net/
[4] http://wiki.fasterxml.com/JacksonHome

6.1.1 Data Model

The figure below describes the data model of the demo app. The mobile app comes with two different views for presenting the consumption data (MeterdataValues) within the selected time interval (15 min values, daily values, monthly values) cumulative, or time series

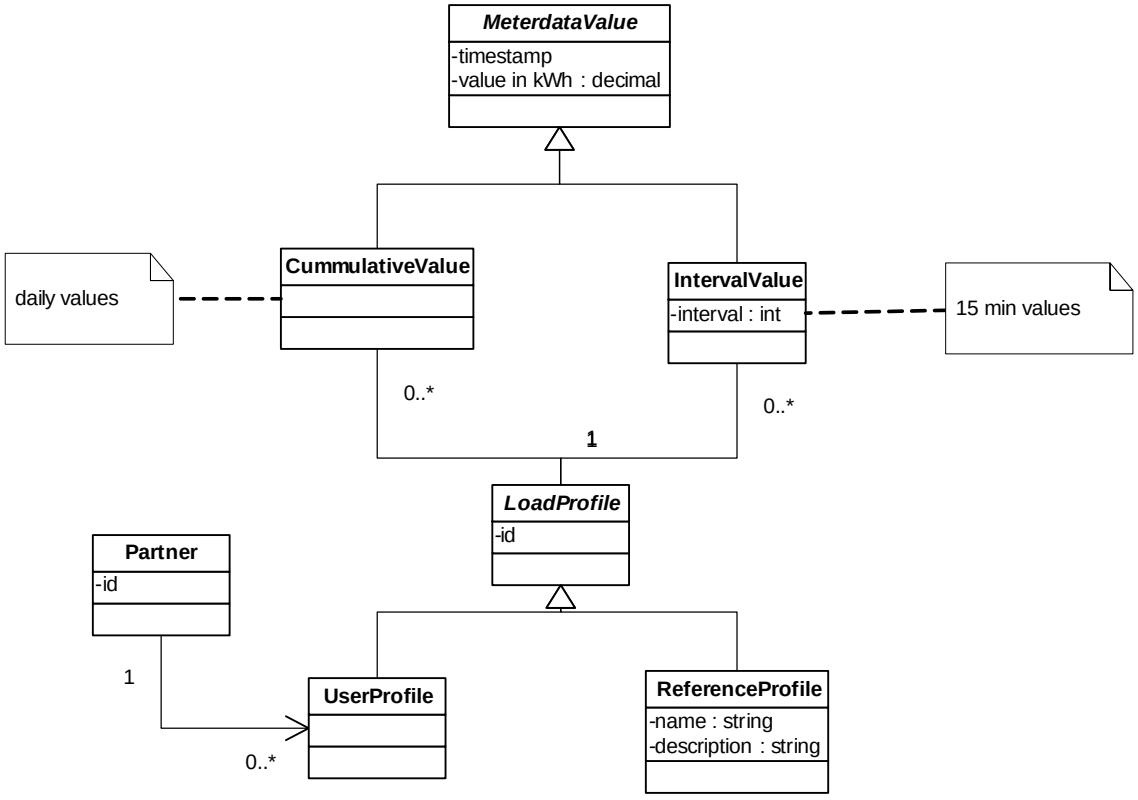

Figure 20 Class Diagram of the Concepts supported by the Smart Meter API

The user profile consists of the credentials which are needed for Authentication and personal settings which have to be stored persistently. The “Loadprofile” entity represents the load profile of the current user and the “ReferenceProfile” represents a third-party load profile which supports comparison functions.

6.1.2 Test API master data

This interface provides information for example about the customer in person, the used device, given feedback. The following table shows view possible examples of application functions in general:

ID	Ressource	Description
1	GET /consumerinstallations	provides master data of a user profile

2	GET /installationdates	provides the installation date of a smart meter
3	GET /declarationofconsents	provides the status of privacy declaration
4	GET /equivalences	provides textual equivalents to consumption values
5	GET /faqs	provides list of frequently asked questions

6.1.3 REST API metering values

This interface provides different functions around energy consumption values of a measure point. The table below shows some examples of possible functions:

ID	Resource	Beschreibung
1	GET /values/<GP>/<id>	provides every consumption value in kWh of a specific measure point
2	GET /referenceprofiles	provides a description of a norm load profile
3	GET /referenceprofiles/values/<id>	provides consumption values for a specific norm load profile

6.1.4 Data accessibility constraints

In general personal data can be only accessed with mobile clients by prior authentication and authorization. The appropriate credentials are passed to the SCA framework. The further process follows as described in Chapter 4.

Beside this common data access mode, the mobile app provides a demo mode. In this demo mode every application function can be tried out without needing any credentials.

The demo function is not implemented as application logic separately. It is realized at the data level. The demo function uses the same application logic as with a default customer. Thus, no additional application logic must be implemented and tested, and the user has access to all of the application functions, available with a full registration, within the demo mode.

6.1.5 Data Security Considerations

The responsible handling of personal data is a major concern of the working group. Therefore the data between the mobile terminals and the SCA platform will be transmitted with SSL encryption at any time.

Downloaded and personalized consumptions values are stored in a secure local database, this is done on the one hand to ensures that the mobile application works without data connection and on the other hand, to ensure that the database is deleted after each logoff, thus ensuring that no user can view consumption data of a different customer.

6.2 Weather Data API

The example weather API provides historical data for cities for a given postal code. The postal code of the smart meter customer can be used to lookup the weather of the current user, who is trying to use the app. The actual data source that will be used for the weather data must yet be determined, mainly because the data can also be an important cost factor for the operation of the App. An alternative would be to use the "OpenWeatherMap"[5], which is an open source weather API for developers.

6.3 Demo APP consuming example APIs

By using a first prototype demo APP it will be shown how a mobile app can be published based on the Smart Citizen Assistant framework and how its unified interfaces can be used. In addition to this functional demonstration the intention of providing a demo APP is to motivate third-party developers to create new applications. In this way, the wide use of the SCA framework is promoted.

6.3.1 Context of the Demo APP

Smart Metering allows electricity network operators to provide detailed energy consumption data for their customers at different online media. It is the central goal of Smart Metering to help customers saving energy. Therefore, current consumption values will be provided on a 15-minute or daily base to raise the awareness for energy usage.

As the figure below shows an entire Smart Metering system consists of intelligent meters (smart meters) at the client side. At the network operator's side transmission technology, computer systems for data acquisition and processing will be needed.

[5] http://openweathermap.org/

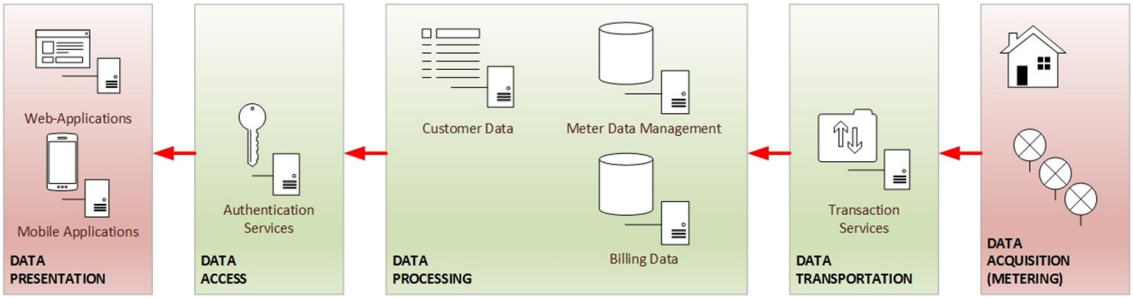

Figure 21 From smart meter to APP- the flow of data.

In the Transform+ research area the new smart metering technology will be implemented, so it offers the opportunity to develop a demo app that will support city-dwellers to find out their energy footprint.

The demo app uses consumption data of electricity smart meters in the Transform+ project area. The application is designed for end user to display their own consumption of electricity in an intuitive manner.

6.3.2 Functions of the Demo App

As mentioned above, the aim of the mobile application is to sensitize the user for the careful use of energy. Therefore, the following functions are provided in the application:

- Simplification of excise representation (creation of awareness)
- Representation of consumption values (easy access to consumption data)
- Comparison of consumption values (between other locations or customers))
- Assistance for energy-saving activities (providing energy saving tips)

Special emphasis is placed to provide comparability between the user's consumption values and different reference points, because it can be assumed that the best way to generate a better comprehension of energy consumption is generated by plausible comparisons. As possible reference points can be viewed (not exhaustive):

- Weather data
- Specific time periods (seasons)
- Attributes of the geographical location
- Living conditions of the end user

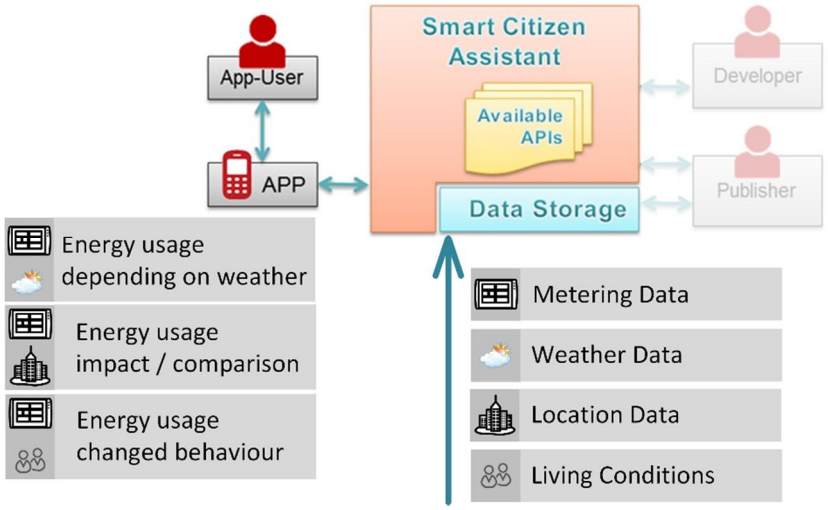

Figure 22 Example usage of data by the demo App.

Accordingly, the combination of energy consumption data with other data sources is of considerable importance to the main objective of the application. The figure below describes the usage of different data sources within the demo app.

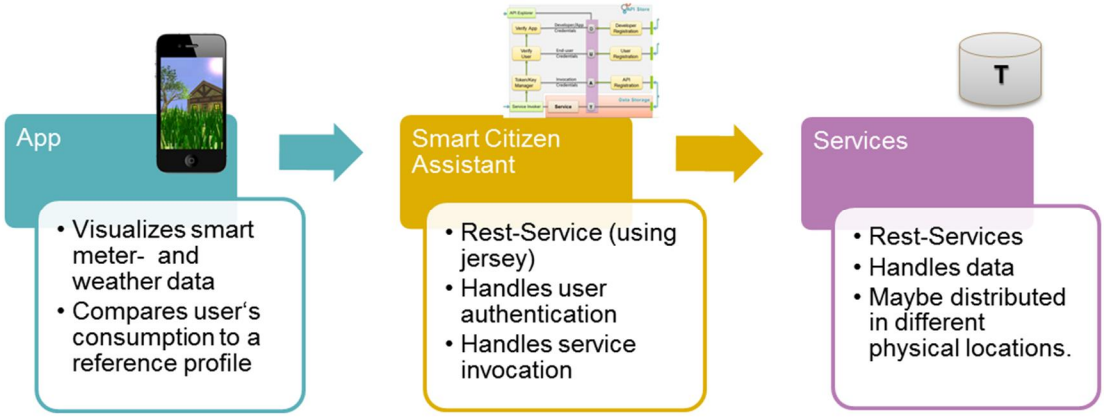

Figure 23 Workflow of the APP accessing the data through the smart citizen assistant.

The Demo APP will be created such that it can show how the smart citizen assistant can help in creating new applications by integrating multiple data sources. The idea is to visualize the impact of energy waste to the environment over a certain time and comparing the data with the weather of the same time period. This gives users a simple tool to compare their energy consumption with a reference profile and the weather.

6.3.3 Internal architecture of the Demo APP

Internally, the demo app consists of a data loader that uses a JSON reader component to load data through the smart citizen assistant. The controller of the application then disperses the data to different visualization components that can modify the "game objects" within the app to visualize the incoming data.

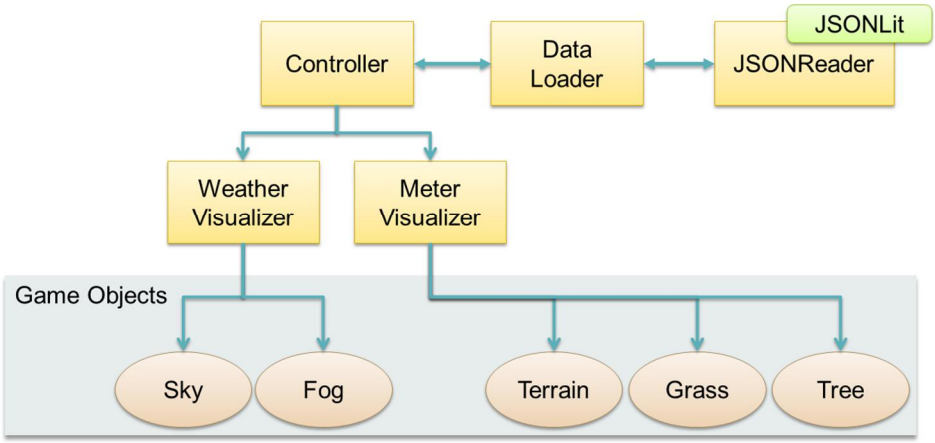

Figure 24 Internal Architecture of the Demo App

In the end, a common paradigm for visualizing multiple sources of data could be defined to create the impression of integrated data visualization on the front end.

Figure 25 Possible "Moods" of the APP visualizing the weather and the energy consumption.