Title:	Document Version:
D3.1 Design and Preliminary Specification of the OESP	1.0

Project Number:	Project Acronym:	Project Title:
FP7-60061-EEB-ICT-2011.6.5	SmartKYE	Smart Grid Key Neighbourhood Indicator Cockpit

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M12 (October 2013)	M12 (October 2013)	R-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Robert Sauter	UDE	WP3

Authors (organization):

Robert Sauter (UDE), Richard Figura (UDE), Lucas Pons (ETRA I+D), Diego García (ETRA I+D), Lola Alacreu (ETRA I+D), José Javier García (BDigital), Deja Ilic (SAP),

Keywords:

SOA, Energy Management, Web Services, Communication platform

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	17.09.2013	New document	Robert Sauter (UDE)
V0.2	14.10.2013	High-Performance Web Services	Robert Sauter (UDE), Richard Figura (UDE)
V0.3	21.10.2013	Complete Design	Robert Sauter (UDE)
V0.4	25.10.2013	Complete Draft	Robert Sauter (UDE)
V0.6	28.10.2013	Peer Review	José Javier García (BDigital)
V0.7	28.10.2013	Peer Review	Deja Ilic (SAP)
V0.8	29.10.2013	Peer Review	Lucas Pons (ETRA I+D), Diego García (ETRA I+D), Lola Alacreu (ETRA I+D)
V1.0	31.10.2013	Reworked according to comments	Robert Sauter (UDE)

Abstract:

Work Package 3 is dedicated to the specification, design and implementation of the Open Energy Services Platform (OESP). The OESP acts as a flexible information hub that decouples the energy applications interfacing different EMS in a neighbourhood – i.e. in the case of SmartKYE both the Business and M&C cockpit – from the heterogeneity of the smart grid and communication infrastructure. The OESP builds on the architecture and services developed and identified in WP2 and provides the necessary infrastructure for the effective processing and delivery of services while allowing the applications to dynamically express their information requirements and taking the capabilities of the EMS infrastructure into account.

This document describes the preliminary specification and design of the OESP. It is closely related to and complements D2.1, which describes the overall architecture of the SmartKYE system and the interfaces provided by the OESP and by the energy management systems. The document consists of three major parts:

First, we analyse the requirements collected in D1.1 for their impact on the specification and design of the OESP. Moreover, we explore the main considerations to enable the deployment in Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) environments.

Second, the core part of the document deals with the specification and design of the OESP with a special focus on scalability. We describe the architecture and the decomposition of the Platform into individual components that may be deployed separately to adapt to the size of the installation. Next, we discuss the node-centric and data-centric communication paradigms as well as the push-based and pull-based access patterns offered by the platform. The combination of these mechanisms offer suitable abstractions for all considered use cases while taking into account efficiency and the potential for optimization of the data flow by the platform. This chapter is completed with the description of the EMS Service Endpoints, and discussions of the publish/subscribe service and grouping service.

The third major part of the document investigates possibilities for high-performance web services. This includes enhancements and optimizations of the different layers of the web service stack spanning HTTP evolutions like SPDY and HTTP 2.0, various compression schemes, and alternative content formats.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose of the Document

The purpose of this document is to provide the preliminary description for the specification and design of the Open Energy Service Platform (OESP). This document is closely related and complementary to D2.1 “Reference Architecture and Energy Services v1.0”. D2.1 describes the overall architecture of the SmartKYE system. Moreover, it contains the definitions of the interfaces between the OESP on the one hand and the clients, energy management systems and other data sources on the other hand.

This document focuses on the internal design of the OESP. It describes the architecture of the platform and the possibilities for decomposition and individual deployment to increase scalability of the overall system.

1.2 Scope of the Document

This document describes the preliminary specification and design of the OESP with the exception of the internal workings of the Complex Event Processing engine which is scheduled for D3.3. It is complementary to D2.1 and both will be updated to describe the final versions in D3.3 and D2.2 respectively.

1.3 Structure of the Document

The rest of this document is structured as follows. In Chapter 2, we review the requirements with respect to the specification of the platform. This chapter is divided into sections that describe different categories of requirements for the middleware. Moreover, we discuss the increasingly important deployment options of Infrastructure as a Service and Platform as a Service. In the following chapter, we describe the specification and design of the OESP. This includes the architecture and the communication concepts supported by the platform. We discuss possibilities to optimize the performance of web services in Chapter 4 followed by the conclusions of the document.

2 Requirements Analysis

The full requirements specification for SmartKYE has been done in WP1 and is described in D1.1, where the requirements are separated into 8 different categories according to the 4 main development parts of the SmartKYE projects and the 4 different EMS types involved.

In this chapter, we shortly discuss the requirements list related to the Open Energy Services Platform (OESP) to provide a framework for the following description. For this purpose we divided the OESP requirements into 6 different groups: core functionality, publish/subscribe, complex event processing, data and information requirements, interface requirements, and performance requirements. In each section, we shortly describe the category and list all requirements that belong in that category. To keep that list compact, the specification table will only list the ID, a short description and the priority. The ID field is unique in the set of all requirements of the requirements analysis in D1.1. The priority field can be from the set: low, medium and high. The full description of all requirements including all further fields is listed in D1.1.

2.1 Core Functionality

This section deals with the core functionality of the OESP: making data available in a uniform way to decouple clients from the set of varied heterogeneous energy management systems and other data sources. Additional core functionalities are:

- support for metadata like timestamps and the origin of information
- data aggregation and KPI calculation
- to enable the management of EMS

2.1.1 Requirements

ID:	OES_006	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should support information summaries.				

ID:	OES_007	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should record the origin of information.				

ID:	OES_008	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should record the timestamp of information.				

ID:	OES_010	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	The OESP should support short-time persistence for data.				

ID:	OES_011	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	The OESP should support parametrization of persistence duration.				

ID:	OES_015	***Type:***	The scope of the product	***Priority:***	High
Description:	OESP should handle both measurements and events				

ID:	OES_024	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	It should provide information from different EMS				

ID:	OES_034	***Type:***	Functional and data requirements	***Priority:***	High
Description:	Health status of services and systems				

ID:	OES_035	***Type:***	The scope of the work	***Priority:***	High
Description:	The OESP should allow EMS to provide data necessary for KPI calculation.				

ID:	OES_037	***Type:***	The scope of the product	***Priority:***	Medium
Description:	The OESP should enable the management of the EMSs				

ID:	OES_038	***Type:***	Performance requirements	***Priority:***	Medium
Description:	OESP should provide information about the health/status of its services and keep historical data of it				

ID:	OES_043	***Type:***	Operational requirements	***Priority:***	High
Description:	Historic data should be kept as necessary for the calculation of KPIs				

2.2 Publish/Subscribe

The publish/subscribe API allows the clients to specify their information needs and receive continuous notifications for both measurements and attribute updates. On the one hand, both the Business Cockpit and the Monitoring and Control Cockpit can access this API to continuously use the information for updates of state information and to adapt management strategies. On the other hand, this is the foundation for a timely delivery of alarms and events.

2.2.1 Requirements

ID:	OES_002	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should distribute the information of the infrastructure efficiently based on the declared needs of the cockpits.				

ID:	OES_003	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow the creation and use of application-defined filters for data.				

ID:	OES_004	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow spatial aggregation of data.				

ID:	OES_006	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should support information summaries.				

ID:	OES_009	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow temporal aggregation of data.				

ID:	OES_015	***Type:***	The scope of the product	***Priority:***	High
Description:	OESP should handle both measurements and events				

ID:	OES_021	***Type:***	The scope of the work	***Priority:***	Medium
Description:	OESP sould support event and alarm management				

2.3 Complex Event Processing

One of the core cross-work-package research topics of the SmartKYE project is the shift of parts of the application functionality into the platform. This fosters the reuse of processing logic and allows the coarse-grained optimization of the data flows by the platform.

Additionally, the requirements describe functionality for spatial and temporal aggregation of data which is not only made available in the CEP engine but also for the on-demand and continuous access to measurement data provided by the core APIs and the publish/subscribe mechanism.

2.3.1 Requirements

ID:	OES_003	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow the creation and use of application-defined filters for data.				

ID:	OES_004	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow spatial aggregation of data.				

ID:	OES_009	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow temporal aggregation of data.				

ID:	OES_005	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should allow shifting repetitive parts of application logic into the platform.				

ID:	OES_041	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	OESP should support persistent creation of processing rules				

2.4 Data/Information Requirements

The category data/information requirements specifies which data and functionality of other information sources – foremost the energy management systems – must be made available through the platform. An important example is the provision of weather forecast information which is both necessary for consumption and productions estimated of most energy management systems.

2.4.1 Requirements

ID:	OES_014	***Type:***	The scope of the product	***Priority:***	Low
Description:	The OESP should provide access to the history of system changes of an EMS				

ID:	OES_017	***Type:***	The scope of the work	***Priority:***	Medium
Description:	Type of Weather Forecasting data that should be handled for WFs: Wind Speed,Wind Direction, altitude,temperature,pressure,location				

ID:	OES_018	***Type:***	The scope of the work	***Priority:***	Medium
Description:	Type of Forecasting data that should be handled Demand:temperature,pressure,location				

ID:	OES_019	***Type:***	The scope of the work	***Priority:***	Medium
Description:	Type of Weather Forecasting information that should be handled for PVs:irradiation, altitude,temperature,pressure,location				

ID:	OES_020	***Type:***	The scope of the work	***Priority:***	Medium
Description:	The OESP should provide access the electrical topology of an EMS				

ID:	OES_032	***Type:***	Functional and data requirements	***Priority:***	High
Description:	Provision of Energy mix				

2.5 Interface Requirements

The category interface requirements contains requests about the interaction of the OESP with other subsystems. The primary requirement is the availability of the OESP via secure web services over the internet. This applies both to the communication with the energy management systems and the interaction with the cockpits. Additionally, issues that affect an EMS should not have any impact on the communication with other systems.

2.5.1 Requirements

ID:	OES_001	***Type:***	Functional and data requirements	***Priority:***	High
Description:	The OESP should support secure web services.				

ID:	OES_013	***Type:***	The scope of the product	***Priority:***	High
Description:	OESP communication with any District Energy Management System (EMS) should be based on web services				

ID:	OES_016	***Type:***	The scope of the work	***Priority:***	Medium
Description:	OESP should have a strong user management mechanism				

ID:	OES_039	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	OESP should support standardized eventing				

ID:	OES_027	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	SmartKYE components system clock shall be aligned to a clock reference				

ID:	OES_031	***Type:***	The scope of the work	***Priority:***	High
Description:	communication via secure web services (over https)				

ID:	OES_028	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	EMS unavailability all site energy subsystems shall not be affected.				

ID:	OES_036	***Type:***	The scope of the work	***Priority:***	High
Description:	The services of OESP should be securely available over the Internet				

2.6 Performance

The main requirements with respect to the performance of the platform are the investigation of how to improve the performance of web services and the need for scalability of the platform. The platform should be adaptable to various deployment sizes to ease adoption with municipalities.

2.6.1 Requirements

ID:	OES_012	***Type:***	Performance requirements	***Priority:***	Low
Description:	The OESP should investigate possibilities to improve the performance of web services.				

ID:	OES_026	***Type:***	Functional and data requirements	***Priority:***	Medium
Description:	OES should have a modular architecture design that assures scalability and adaptability to different deployment sites.				

ID:	OES_042	***Type:***	Performance requirements	***Priority:***	Medium
Description:	OESP should be scalable and high-performant				

ID:	OES_033	***Type:***	Functional and data requirements	***Priority:***	High
Description:	Aggregated Energy data available every 15 minutes				

2.7 Cloud-Aware Design

In recent years, “cloud-*” has become widely known buzzword both in the consumer and enterprise world. More specifically, the services offered by cloud computing providers can be grouped into several categories. The most important categories (c.f., Figure 1) for SmartKYE are infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). While the cloud designator has been overused, the potential to use the OESP with these service categories significantly extends the flexibility for deployments.

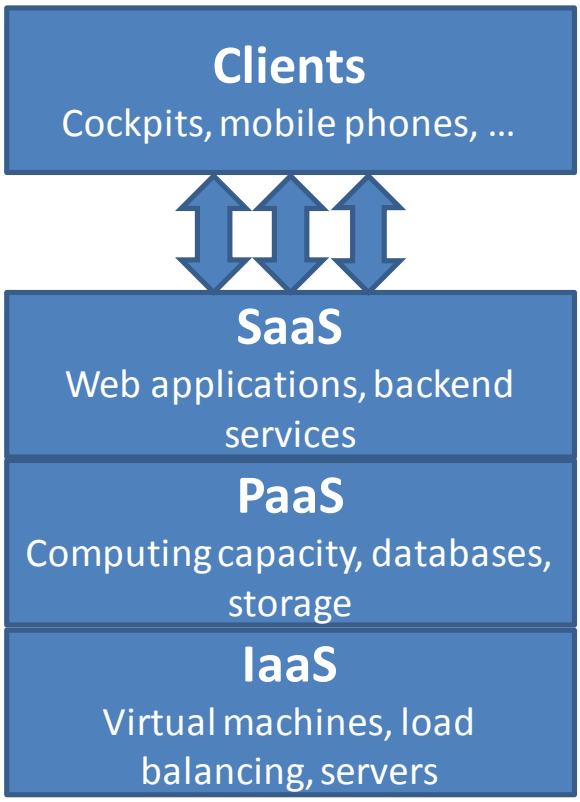

Figure 1: Cloud Computing Service Models

2.7.1 Software as a Service (SaaS)

Software as a Service is as much a business model as a technological concept. From a business perspective the most important difference is that instead of buying Software clients instead purchase the access to software. Often called “on-demand software”, pricing can vary significantly and includes pay-per-use and pay-per-user usually involving a subscription component.

From a technological viewpoint, this model is most prevalent as a web application but can also include, e.g., traditional remote desktop access. A main difference is that the cloud provider manages his infrastructure and is responsible for providing the desired or agreed-on service level.

While the technological paradigm is already implied for the OESP, since the core capability is the communication with systems over the internet, it is important for the project to not limit the possibility of both business aspects: the management and operation of the platform may both be handled by a dedicated service provider or for example by the municipality itself.

2.7.2 Platform as a Service (PaaS)

While SaaS is mostly targeted at users of the software – both in the consumer and enterprise area, Platform as a Service is a paradigm for the developers of applications that run in the cloud. In this model, cloud operators deliver a computing platform on a certain abstraction level, usually a programming language execution environment, e.g., the Java Virtual Machine. Often the access to the resources of the operating system is severely limited and instead the operator provides dedicated APIs for example for access to persistent storage. The abstraction level may vary widely and spans file based, object based and various database paradigms.

The core challenge tackled by the cloud operator is the seamless scalability for the

application developers. This is usually targeted at request-response based web applications that coupled with a machine spanning persistent storage model allows the transparent execution load balancing on a large number of physical machines.

While most web applications already drop the assumption on the permanent execution of the software, most commonly the major challenge in developing for a PaaS cloud is the limitation of the storage API that at least requires design considerations to enable optimal scalability.

2.7.3 Infrastructure as a Service (IaaS)

A more basic offering of cloud service operators is the Infrastructure as a Service model. This approach offers computers (physical or virtual machines) to the application developers. These can install a variety of operating systems on them and are usually responsible for managing and updating the machines. The main difference to traditional offerings is that the operating system image can be arbitrarily replicated to a number of machines. This can also be done on demand, e.g., to handle load peaks or adapt the capacity based on periodic schedules (e.g., day vs. night).

Additionally, the cloud operator usually offers additional services. This can include load balancing, firewalls, and network configurations but also pre-defined operating system or application server images. Another area is distributed storage and backup both on the traditional file abstraction level and also on block or object level.

2.7.4 Conclusions

Both Platform as a Service and Infrastructure as a Service are interesting deployment scenarios for the OESP. Therefore, the following main challenges have to be addressed in the design:

- no permanently running software (periodic tasks are possible)
- strong focus on single request-response operations
- minimal dependency on global persistent knowledge
- abstraction layer between core functionality and persistent storage services

These design considerations are also fully in line with scalability as a major design goal.

3 Specification and Design

This chapter describes the internal mechanisms of the OESP. As such, it must be seen in combination with D2.1 which contains the overall architecture of the SmartKYE system and the definition of the interactions between the platform on the one hand and the energy management systems, the cockpits and other interaction partners on the other hand.

3.1 Architecture

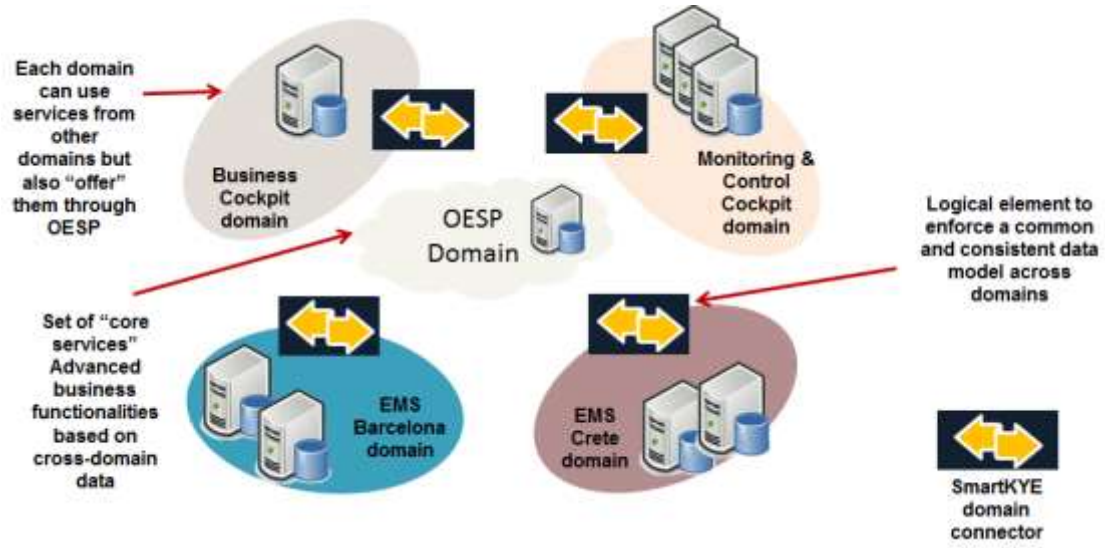

Figure 2: SmartKYE Federated Architecture

In Figure 2, we show the high-level architecture view of the SmartKYE system. Logically, the OESP is the single communication hub that enables the interaction among the entities in the federated SmartKYE system. However, a physical design following this architecture would include a single point of failure and limit the scalability of the system considerably.

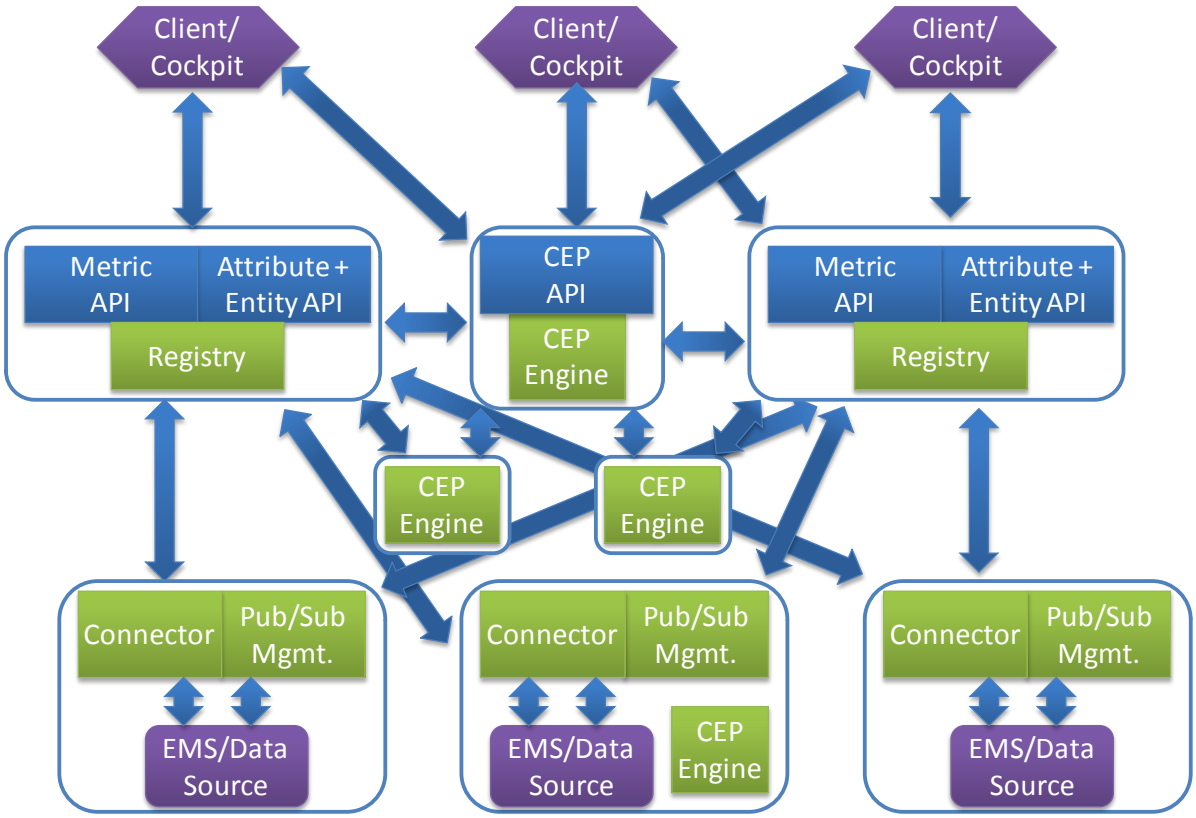

Figure 3: Example Physical Architecture (OESP components in green and blue)

In Figure 3, we show an example for a possible physical deployment of the SmartKYE system. The example highlights several of the following key design decisions of the OESP:

- Potential separation of the OESP APIs (although we expect the entity, metric and attribute API to be usually co-located)
- Multiple OESP instances to balance the load of clients
- CEP Engine separated from CEP API
- Multiple CEP instances
- Usually: physical co-location of EMS, connector and publish/subscribe system

To simplify the figure, we do not show the grouping service, which is discussed in more detail below.

The entity, metric, and attribute API all use internally a registry that contains information about the EMS present in the system. It is noteworthy, that this registry does not contain information about all entities, but just about the endpoints. Entities can be queried on demand from the EMS but also cached for to improve efficiency.

Similarly, the metric management component and the publish/subscribe components contain caching functionality. However, this is transparent to both the clients and data sources and, thus, allows the parallel development and optimization of the system.

In general, the OESP does not store information itself but requests this information from the appropriate energy management systems. This improves scalability – as with adding an EMS automatically the necessary storage space is added –, flexibility – as the EMS can decide when and where to store information –, and efficiency – as only the information that is actually required by any clients has to be transferred from the EMS while still allowing the OESP to use caching to exploit that some information will be requested by multiple clients.

If a request spans multiple EMS, the OESP combines and potentially aggregates the

individual responses from the EMS. If one or more energy management systems do not respond at all or return errors, the OESP will return the partial information received together with a description of which data sources were not available at all and the error information received from the data sources.

3.2 Communication Paradigms

The platform is a communication and unification layer that manages the interactions between the different systems in the envisioned scenario. In SmartKYE the applications (e.g. for monitoring, prediction and management) have to deal with information coming from a vastly heterogeneous set of different energy systems and devices (e.g., wind farms, public light controllers). The OESP with its adapters enables uniform data access and cooperation between the applications and abstracts from underlying differences.

Communication solutions can be distinguished into different types depending on the communication mechanisms they provide. One major division is between data-centric and node-centric solutions. Mostly orthogonal to this distinction is the use of an on-demand (pull) or continuous notification approach (push). We describe these major categories shortly in the following. Our platform supports all paradigms to allow the application to choose the solution suitable for a given task.

3.2.1 Node-centric Communication

In a node-centric communication the most important task is to get information from a specific node (e.g., a certain entity) within the network. In this case the communication has to provide an API to address and access a certain node to read/write data or access services. This kind of communication is in common use and examples include CORBA or Java RMI.

The advantage of this approach is the direct control of individual nodes which is especially suitable of the execution of commands. While often directory services are used to find the nodes that provide a certain service, the relatively strong coupling between entities can limit the advantages resulting from a fully distributed system.

To access data from a number of nodes, a consumer has to find the nodes (e.g., using a directory service) and request the data from each one individually.

3.2.2 Data-centric Communication

In a data-centric communication, the key concept is the data itself. The members of the network can be separated into providers of data and consumers specifying their interest in for certain types of data. This separation is not strict as individual nodes can contain both components providing data and consuming data.

From a logical viewpoint, the key point for the user of the communication is to simply specify an interest for data without first having to locate the relevant devices and query each of them individually. Likewise, providers of data just advertise the kinds of data they provide. The platform is then responsible for matching subscribers and publishers and establishing the necessary communication connections. This loose coupling enables the easy creation of fully distributed systems.

From a network viewpoint, the key advantage is the possibility for optimizations by the communication. Depending on the network topology, QoS requirements and the provided communication technologies, the communication can select and adapt suitable communication protocols to efficiently distribute the data.

Push-based data-centric communication solutions allow data to be typed. Through this it is possible to connect consumers to a certain type of data (e.g. energy-readings). Every consumer can so decide which type of data it wants to receive and every producer can

decide which type of data it offers.

3.2.3 On-Demand (Pull)

The on-demand paradigm allows application to request data in the moment it is required, e.g., to generate a user-requested diagram. On the one hand, this allows specifying directly and exactly which information is needed and only this information has to be transferred over the network. On the other hand, the need to first issue the request before the involved systems (e.g., various EMS) start to process the request followed by communicating the response implies a certain minimum delay which often varies depending on the complexity of the request and the accessed systems. In general, this approach is especially suited for interactive applications interfacing directly with the user.

3.2.4 Continuous Notifications (Push)

Push-based systems allow applications to specify an interest in data updates which are then automatically transferred from the data sources to the intended recipients. Especially periodic requests known in advance (e.g., energy consumption every 15 minutes) can be efficiently scheduled and combined to achieve better overall throughput of the system. Additionally, since the request is already known in advance, the delay between the time information is available at the source and is transported to the consumer can be reduced. This approach is well suited for most monitoring and management application that required regular periodic updates of the information of the subsystems to act in a timely manner based on their predefined behaviour specifications.

3.2.5 Comparison

Considering node-centric and data-centric communication, both approaches are viable solutions for both accessing groups of nodes and individual nodes, but each solution requires more work for the user and limit optimization capabilities in the network when applied to the “wrong” use case.

In SmartKYE the data of the energy management systems are much more important to enable functions like monitoring or prediction than the individual devices themselves which is an argument for a data-centric approach. However, there is also a need to access individual devices for example to change configuration parameters or control settings, which is best approached by a node-centric approach. To provide maximum flexibility and ease-of-use for the user while preserving the capability for optimizations, the SmartKYE communication supports both approaches.

Similarly, there are good reasons for both pull and push communication. On the one hand, the interactive features of Business Cockpit can best be served with on-demand requests for data. On the other hand, a significant part of the Management and Control Cockpit deals with always-on functionality and can make good use of the pull paradigm.

While largely orthogonal concepts, one important combination is the publish/subscribe paradigm. With a publish/subscribe mechanism every producer of data does not send its data to consumers directly, instead the producers only offers the data for publishing. Every consumer that is interested in a certain set of data can subscribe to these publications. As soon as new data is published, it is sent to all the subscriber of the data for further processing (cf. Figure 4).

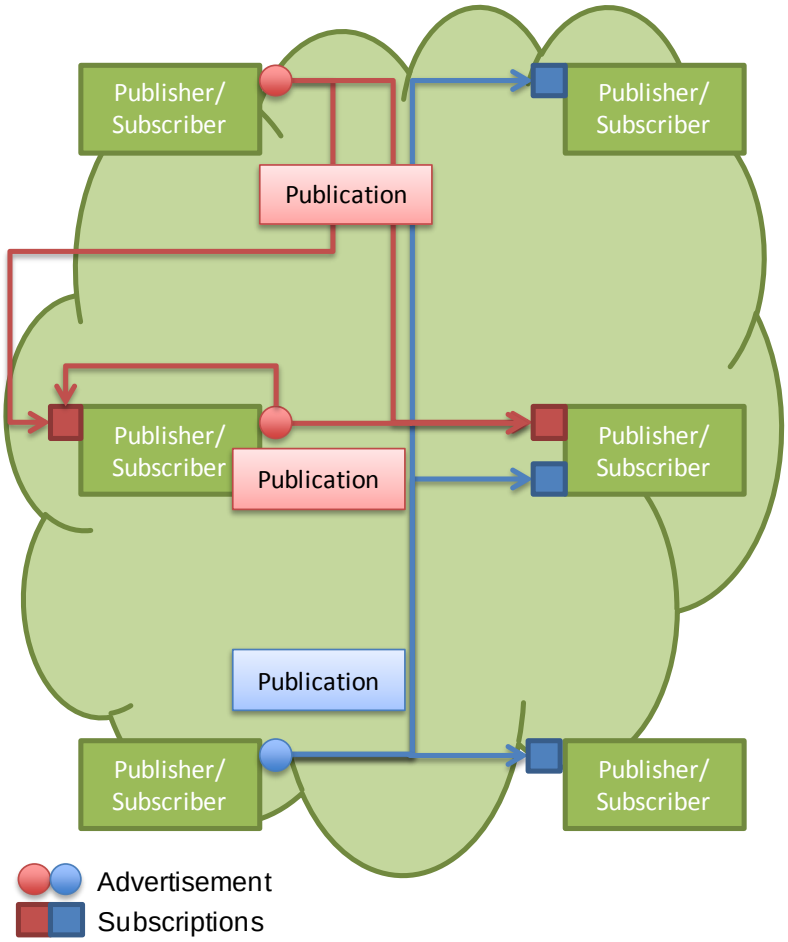

Figure 4: Publish/Subscribe System, Logical View

To make it easier for subscribers to receive a certain set of data, the publish/subscribe mechanism supports filters. With this every data can be published to a certain channel. Every subscriber of a channel will receive every data of this channel as soon as it is available. In addition to channels (sometimes called topics) that identify the type of data, we also support the specification of periodic updates. In the context of SmartKYE, the metric types and the attribute types are the implicit channels that consumers can subscribe to.

3.2.6 API Mapping

The different communication paradigms are mapped to the API in two ways: the support for both data-centric and node-centric communication of provided by the flexible nature of the EntityFilter (c.f., D2.1) structure which is used in almost all API calls to specify the targets of the call. This structure supports both the specification based on entities and their relationships, i.e., node based selection, and the data-oriented selection based on which metrics or attributes an entity provides.

Second, both the metric and attribute APIs provide functions to subscribe to notifications. These complement the function to request information on-demand and, thereby, provide for both push and pull based access to the data of the EMS. Likewise, the CEP engine supports both the on-off execution of a processing graph and running it as a continuous operation.

3.3 EMS Service Endpoint Description

As described in D2.1, most services of the OESP are mirrored by the energy management systems: the entity, metric and attribute APIs are also implemented by the connectors that interface with the EMS. Only the publish/subscribe mechanism are differently implemented as described in the next section.

From the perspective of the platform, an EMS or any other data source is responsible for one or more root entities and their children. However, to increase the flexibility, the OESP allows distributing the responsibility for different metrics and attributes among various data providers (c.f. Figure 5). This would for example allow an ESCO to provide price and cost information for its clients or to outsource the prediction of production to dedicated service providers.

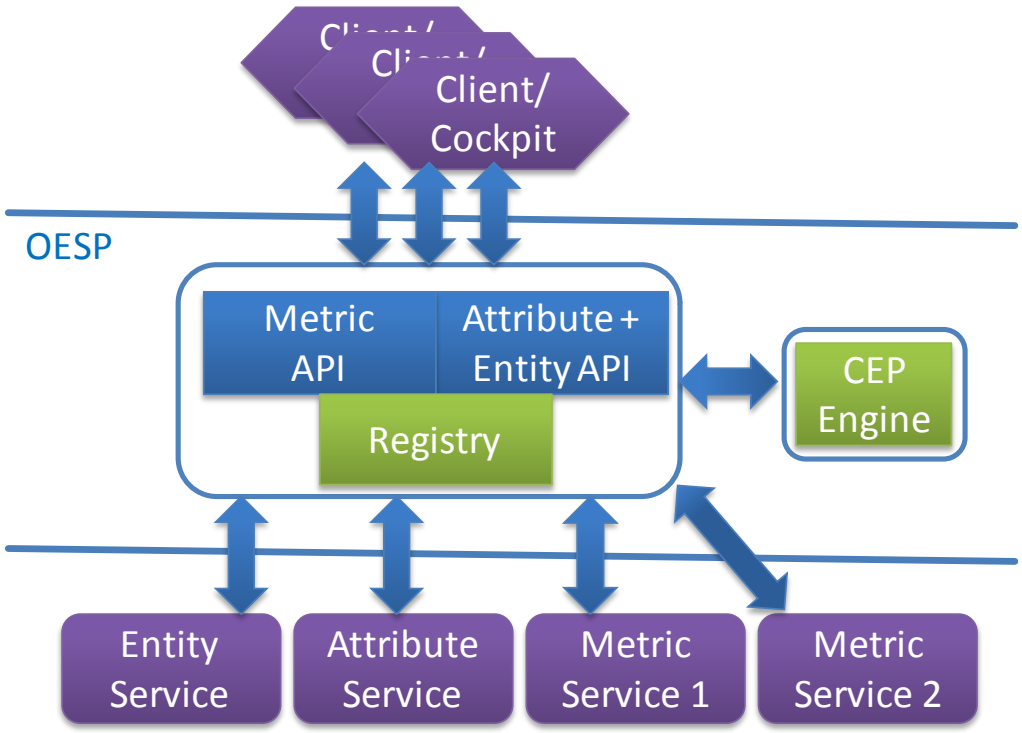

Figure 5: EMS Service Endpoint Description (OESP components in blue and green)

Additionally, the OESP allows specifying that a CEP processing graph is used to provide the information for certain metrics. This allows the reuse of common processing rules for example to calculate the cost based on price and consumption information from other servers or to forecast energy consumption based on historical information.

Moreover, all services can of course in turn make use of all platform APIs to provide the requested data.

By using the CEP services either directly or indirectly, the information providers can benefit from distributed nature of the platform and make scale the use of CEP engines based on the current demand.

In addition to specifying the functional endpoints, the EMS also specifies the authentication requirements. On the one hand, for complete flexibility that allows integrating authentication and authorization in the existing infrastructures of the different institutions, the EMS may specify an OAuth 2.0 Authorization Server (Internet Engineering Task Force (IETF), 2012) that will be consulted for requests by any clients. On the other hand, to simplify EMS implementations, they may also rely on the user authenticated by the OESP.

3.4 Publish/Subscribe Service

For the publish/subscribe service, there are two different use cases to consider. For metrics, we assume clients to specify their interest for periodic updates of (potentially aggregated) measurements. This does not require any special support by the EMS as the OESP can schedule its operation to query this information on-demand and – considering the information needs from multiple clients – efficiently distribute this information.

For the attribute service, however, the clients request notifications about update of state changes. These state changes can occur at any time and, therefore, the EMS must provide additional functionality to notify the OESP about these changes. However, to simplify the complexity for the EMS considerably, the logic when to forward these updates to which clients, is encapsulated in the OESP. To reduce the communication, we assume that in most cases the management functionality will be co-located with the EMS together with the connector for the OESP.

3.5 Grouping Service

The grouping API is omitted from the previous figures to simplify the illustrations. While similarly independent as the metric, entity and attribute service, we nevertheless expect to co-locate these services for all deployments.

In Figure 6, we show the API of the entity grouping service as defined in D2.1. The grouping service is special with regard to the need for global information: the defined groups are available at all platform instances. However, by omitting the functionality to change groups, we can severely reduce the consistency requirements imposed by this service. Instead of forcing a synchronous update of groups to prevent errors that use wrong definitions, this approach focuses on eventual consistency. Whenever a group is used that is not currently known to the instance of the platform, the group service is tasked with resolving this group. Depending on the implementation this may involve accessing a central authoritative service, a peer-to-peer resolver, or a distributed storage system.

When a group is deleted, this information is propagated to all OESP instances. However, the worst case is that a group can be used for a longer time than intended, but it is impossible that a wrong definition of a group is used.

In any case, a group definition can contain an arbitrary list of entities spanning multiple energy management systems.

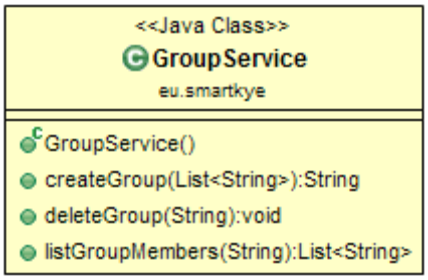

Figure 6: Services for Entity Grouping

3.6 Conclusions

The primary non-functional considerations for the OESP are the federated nature of the SmartKYE system and the goal to seamlessly scale arbitrarily sized deployments.

Both considerations influenced the API specification to consider smart EMS as the primary interaction partners and to largely mirror the APIs between the platform and its clients and between the platform and the data sources.

To tackle scalability in general and the suitability for Infrastructure as a Service and

Platform as Service deployments, the design considers a fully distributed system with potentially multiple instances of the platform core services and CEP engines. These systems rely on a minimum of shared state to increase their isolation and, therefore, the scalability of the system.

4 High-Performance Web Services

In this section, we discuss alternatives and enhancements for the layers of the web service stack to increase the efficiency and performance of the interactions between the OESP and its clients and data sources.

4.1 HTTP 1.1 Alternatives and Developments

While HTTP enjoys almost universal adoption, primarily the evolution of web applications involving more data and requiring reduced latencies have spurred the search for alternatives in recent years. To guarantee operability we do not consider the development of new solutions but instead plan to follow closely the evolution of the approaches described below.

4.1.1 SPDY

SPDY (The Chromium Projects) is an application-level protocol aiming primarily at reducing latency for web applications. It uses the following approaches to reduce latency and bandwidth requirements compared with HTTPS:

- Use one TCP connection for multiple request-response interactions
- Parallelize multiple requests to alleviate the impact of long delays
- Compress headers
- Omit identical header between multiple requests
- Require content compression and encryption
- Allow communication initiated from the server

For the OESP, the most relevant advantage is the header compression as for usually small messages, e.g., notifications of the publish/subscribe mechanism, the HTTP header incurs a very significant overhead.

4.1.2 Microsoft S+M

Microsoft S+M (Microsoft) is based on SPDY and follows similar goals. In addition to integrating the framing from WebSockets, this approach considers resource constrained mobile services and reduces the requirements of SPDY for CPU-intensive operations like encryption and compression.

4.1.3 QUIC

In contrast to the previous approaches that are based on TCP and provide more evolutionary approaches to improve HTTP, QUIC (Roskind, 2013) is based on UDP and tries to further reduce the delay incurred by TCP. However, in contrast to SPDY, QUIC is a more recent and experimental effort and does not yet enjoy the adoption of SPDY of third party applications and servers.

4.1.4 HTTP 2.0

In the last year, the IETF started the process to develop the successor to the HTTP 1.1 standard from 1999. After considering multiple proposals including SPDY and Microsoft S+M, the committee settled on SPDY as the starting point. Currently, the IETF is planning for winter 2014 for submitting the draft proposal while a first version has already been published to facilitate the evaluation of real implementations.

4.1.5 Conclusions

Since SPDY already enjoys relatively widespread adoption and several mature

implementations are available, and it is the foundation for the planned HTTP 2.0 standard, we plan on evaluating the system with SPDY. Depending on the availability of the other approaches in the next year – foremost HTTP 2.0 – we will also investigate alternatives for the evaluation.

4.2 HTTP Content Compression

While the important alternatives to HTTP feature compression not only of the content but also of the headers, their use is still limited to selected implementations. HTTP content compression, however, is almost universally available and part of the HTTP content negotiation. While various compression algorithms are available, gzip is the most widely implemented scheme.

Since gzip is based on the compression of repeated byte sequences, it is well suited to reduce the overhead of XML tags and we expect its use in the complete system.

4.3 XML Specific Compression

In contrast to the lossless HTTP content compression algorithms that are generally applicable to all data, there are also several approaches to exploit the knowledge of XML syntax and schemas to reduce the size of the exchanged messages.

4.3.1 Fast Infoset

Fast Infoset is a standard by both ISO and the ITU-T (ITU Telecommunication Standardization Sector) for reducing both the size and processing overhead of XML documents. It exploits the syntax of XML and based on ASN.1 (c.f., below) provides a much more efficient encoding.

4.3.2 Efficient XML Interchange (EXI)

Efficient XML Interchange is a recommendation of the W3C with a similar goal as Fast Infoset. Compared to FI, it also uses information from the XML schema associated with an XML document to improve the encoding. While this requires the schema to be known both by the encoder and decoder, this does not pose a significant problem for the use in the SmartKYE project as the data model is known by all entities involved.

4.3.3 Conclusions

While both discussed approaches promise not only reduced size but also reduced processing time at least for the decoder, their adoption and the availability of implementations is very limited. There also does not seem to be a clear favorite of the solutions. Therefore, we are currently planning to follow the development in this area and depending on the results using the other solutions – foremost the HTTP content compression – decide in the performance optimization phase the need for evaluating one of the proposals.

4.4 XML/SOAP Alternatives

While the previous section discuss the replacement or enhancements of the different layers of the SOAP-over-HTTP stack, in this section we consider complete alternatives. In the last years, the RESTful architecture has gained attention mostly in the area of web applications as the communication mechanism between browser and server. This approach is based on using the HTTP verbs GET, POST, PUT and DELETE to implement CRUD-oriented (create, read, update and delete) APIs.

Compared to SOAP, this reduces the overhead incurred by the SOAP envelope considerably. Additionally, it allows using arbitrary formats for the content of the requests and responses. We shortly discuss three alternatives to XML in the following.

4.4.1 Javascript Object Notation (JSON)

4.4.1.1 Overview

JSON (JSON) (Aziz & Mitchell, 2007) is a lightweight data exchange format specified by Douglas Crockford in 2002. By now it is described in RFC 4627. The goal to develop this language was to have a very simple, easy to read/write or parse/generate data exchange format in the JavaScript literal object notation. The language was first called JSML (JavaScript Message Language) later the name was changed due to a name conflict to JavaScript Object Notation.

JSON displaced XML in some areas as data exchange format. It can be used in Ajax instead of XML; Yahoo uses it for some Web services (since December 2005) and Google offers JSON feeds for its GData web protocol (since December 2006).

Compared to XML as another data language, JSON is much easier to read and write because there is no need for tags.

4.4.1.2 Language

JSON is a subset of JavaScript literal object notation by design (ECMA 262 3rd edition 1999). That means JSON can be parsed by JavaScript implementations very easy.

Since JSON is derived from JavaScript their syntax are almost the same. In fact every JSON data can be parsed with JavaScript using the “eval” function. For example for parsing JSON data directly into a JavaScript Object:

```
var myObj = eval("(" + JSON-Data + ")");
```

Due to conflicts with reserved JavaScript keywords, JSON has much stricter rules for literate values. For example the name of an object member in JSON has to be a valid JSON string and so has to be enclosed by quotation marks.

JSON builds on two structures that are seen as the intersection of the most modern programming languages:

1. Collection of name/value pairs like objects, records, structs, hashes, property lists. From now on only called objects.
2. Ordered List for values like arrays, vectors, lists. From now on only called arrays.

JSON Objects are unordered sets of name/value pairs. Objects are enclosed by curly brackets. Names and values are separated trough a “:” and the name, value pairs themselves are separated by “,”.

JSON Arrays are ordered collection of values. Arrays are enclosed with closed brackets. Values inside the array are separated by “,”.

The definition of the implied structures:

JSON names or strings are collections of Unicode characters enclosed by double quotes.

JSON values can be strings in double quotes, numbers, Boolean values, null, objects or arrays. These structures can be nested.

JSON numbers are like C or Java numbers but it is not possible to define them in hexadecimal format.

4.4.1.3 Encoding

There is no special encoding for JSON data. This means the data will be transmitted as

plain text. But JSON data needs less space written to a file (and so for transmitting) compared to XML. That is because there is no use for tags which results in less overhead compared to XML.

There exist approaches for a binary encoded serialization on top of JSON like BSON (Binary JSON (BSON)) or BISON (Binary Interchange Standard and Object Notation (Jäger, 2007)). In these approaches the data type of a JSON stream is binary encoded, but the value itself is not. Because of the plain text values these approaches have nearly no advantage over JSON in their consumption of space.

4.4.2 ASN.1

4.4.2.1 Overview

ASN.1 (Abstract Syntax Notation One) is a notation used for describing data structures and their physical representation for transmitting in a programming language independent way (ITU) (The ASN.1 Consortium, Inc., 2003). The first work for developing ASN.1 began 1982 in the International Telegraph and Telephone Consultative Committee (CCITT, French acronym). In 1984 ASN.1 became an ITU-T ASN.1 Standard and in 1986 an ISO standard. The cryptic name (ASN dot 1 instead of ASN1) was chosen to avoid confusion with ANSI – the American National Standards Institute.

The ISO 8824 standard is split into four parts:

- ISO 8824-1 | ITU-T X.680: Specification of basic notation.
- ISO 8824-2 | ITU-T X.681: Information object specification.
- ISO 8824-3 | ITU-T X.682: Constraint specification.
- ISO 8824-4 | ITU-T X.683: Parameterization of ASN.1 specifications.

In ASN.1 encoding and data definition is done in two separate ways. There also exists the ISO 8825 standard for encoding this involves amongst others:

- ISO 8825-1 | ITU-T X.690: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
- ISO 8825-2 | ITU-T X.691: Specification of Packed Encoding Rules (PER)
- ISO 8825-3 | ITU-T X.692: ASN.1 encoding rules: Specification of Encoding Control Notation (ECN)
- ISO 8825-4 | ITU-T X.693: ASN.1 encoding rules: XML Encoding Rules (XER)

ASN.1 also allows the specification of custom encoding for physical representation. This can be done through an independent language: Encoding control notation (ECN).

ASN.1 is nowadays a widely used notation applied in many different applications, organizations or projects like AT&T, Intel, IBM, Microsoft, 3COM, American Express, GTE, MasterCard, VISA for Telephony, Audio & Video over the Internet, Manufacturing, Network Management and so on.

4.4.2.2 Language

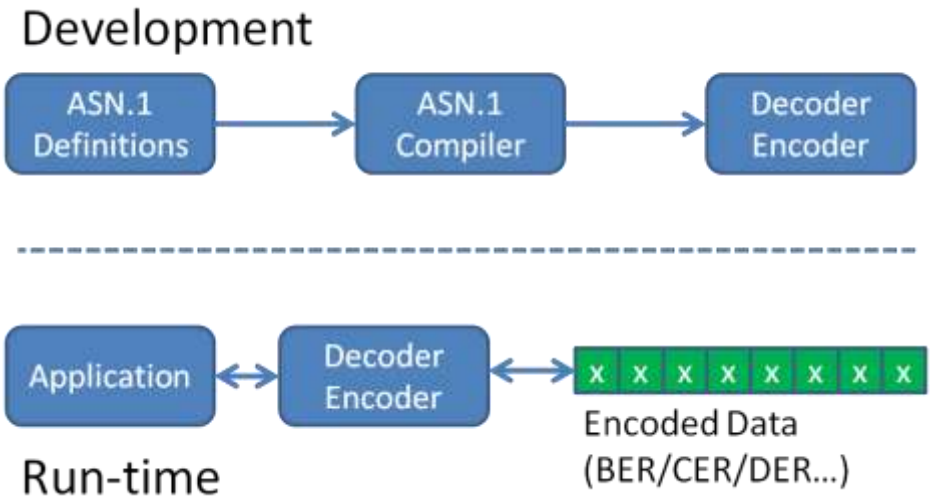

Figure 7: ASN.1 Elements

ASN.1 allows specifying the information abstraction as well as the encoding for data. The information abstraction is done through abstraction syntax. The encoding is done through encoding rules that translates the abstraction syntax into transfer syntax.

ASN.1 has four classes of data types

- UNIVERSAL
  This class is restricted to the ASN.1 built in types. All types of this class have to be distinguishable from all other data types.
- APPLICATION
  This class represents user specific data types that are widely used within a certain context. These types are usable by a set of applications.
- PRIVATE
  This class represents user specific data types that are defined for the use in organizations or by countries for private use.
- CONTEXT-SPECIFIC
  This class represents further user and context specific data.

ASN.1 allows type and value definition. The assignment is done through ":==". The following data types can be used as UNIVERSAL data types:

Basic types:

- BOOLEAN: True or False
- INTEGER: boundless size
- ENUMERATED: Custom values
- REAL: x*y^z with y = [2, 10]
- BIT STRING: String of bits. The length is unbounded
- OCTET STRING: Binary data. Length is a multiple of eight.
- NULL: Null
- PrintableString: String with only printable characters
- UTF8String: String of UTF8 characters
- OBJECT IDENTIFIER: Name information objects

Complex types:

- SEQUENCE: Sequence, different types
- SEQUENCE OF: Sequence, all the same type

- SET: Set, different types
- SET OF: Set, all the same type
- CHOICE: Specify a collection of distinct types from which to choose one type

Subtype Declaration:

ASN.1 allows subtype declaration. It is possible to specify certain values directly as well as a range of values using “SIZE”.

- For single Values:
  - oneTwoThree ::= INTEGER (1 | 2 | 3)
- For sets of values (INCLUDES):
  - oneTwoThreeFour ::= INTEGER (INCLUDES oneTwoThree | 4)

Definitions of types and values can be grouped together in “modules”. Each module is an unordered set of related definitions with unique identifier. A module consists of: The name of the module, identifier (sequence of non-negative numbers), “:==” and the body of the module surrounded by BEGIN and END.

4.4.2.3 Encoding

Encoding is done separately from the data specification within the abstract syntax. Encoding can be done through predefined encoding rules, or with custom specifications. The most encoding rules use the TLV (Type-Length-Value) approach to encode the data (Within ASN.1 TLV is also known as ILC: Identifier-Length-Content).

TLV Approach:

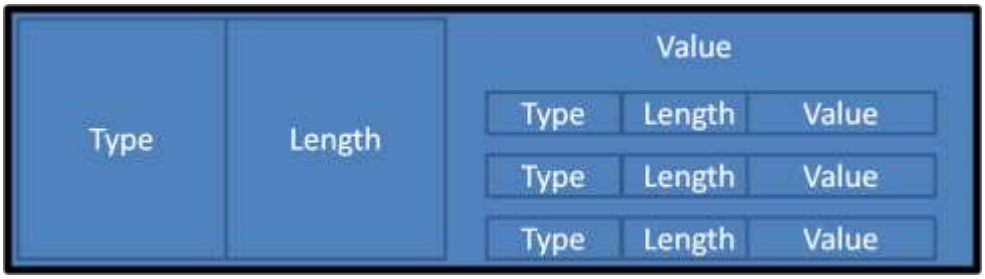

Figure 8: TLV Encoding

With the TLV Approach every data element is separately encoded.

- Type: The type of the encoded element.
- Length: The length of the encoded element.
- Value: The value of the encoded element. If the encoded element is a complex type the included elements also have type length and value fields. This means recursion is possible.

Functionality of Predefined Encoding Rules (BER, PER):

BER (Basic Encoding Rule):

BER is a very simple encoding rule for ASN.1 using the TLV approach. The TLV type-field of 8 Bit is separated into three different sub-fields, specifying the TLV-value:

- 2 Bits class type, possible values: UNIVERSAL, APPLICATION, CONTEXT-SPECIFIC, PRIVATE
- 1 Bit type complexity, possible values: primitive or complex types

- 5 Bits sub type, possible values for UNIVERSAL subtypes are: Boolean, Integer, Bit-String...

Length:

The Length field (8 Bits) specifies the length of the data of the value field. BER supports three different ways to save the length of the data.

Short Form:

The length field starts with “0”. There are 7 Bits left to specify the data length.

Long Form:

The length field start with ”1”. 7 Bits are used to specify the count of byte-blocks that will be used to specify the length of the data (0, 127 reserved and cannot be used for this purpose). This means the length field could be up to 126 Byte, which means there are 1008 Bits to encode the data length. This allows a data length up to 2,74306*E303 in decimal.

Indefinite Length: starts with 10000000 the end of the data is specified through two 8 Bit-blocks with the containing only zero bits.

Value:

This field encodes the data of the type. This is also given in 8 Bit blocks. To represent a Boolean false for example this fields has 8*0

This means for example, the full encoding for a Boolean value needs 3 Byte: One Bit type, one Bit Length and one Bit data.

PER (Packed Encoding Rules):

PER is another encoding rule for ASN.1 that targets small encoding size. TLV is not used in this approach. Instead the type of the data is only given if it is necessary (but length and value fields are always used). If the type can be discovered by context, the type field will be dismissed. Another distinguishes between BER and PER is that PER do not use a fixed field size of 8 Bits. The length-field is smaller if less than 8 Bits are needed to specify the length of the data. Also the size of the value field has a variable length.

There are many more encoding rules for ASN like XER (XML Encoding Rule) or GSER (Generic String Encoding Rules) for human readable formats.

ASN.1 also allows specifying custom encoding rules with ECN (Encoding Control Notation):

ECN is developed and maintained by ITU-T as ITU Recommendation X.692 (ISO/IEC 8825-3). With ECN it is possible to design a complete new set of encoding rules for ASN.1 data types, or to overload/specialize encoding rules from existing definitions like BER or PER.

4.4.3 Protocol Buffers

4.4.3.1 Overview

In 2008 Google released “Protocol Buffers”, a solution for serializing and deserializing of objects for transporting data over networks or storing them (Google). Protocol Buffers have been used internally at Google since 2001 and the revised version 2.0 was the first to be published. Protocol buffers message types are defined in text files using a simple language and code generators produce the necessary code for serializing and deserializing objects of that type. Google cites forward and backward compatibility and small data size (especially compared to XML) as primary goals for the development of their solution required by the

large number of systems and incremental updates on them in their data centres.

4.4.3.2 Language

Protocol buffers message types are defined in simple text files with the ".proto" extension. To some extent, the language is related to structs in C: a message (comparable to a struct) consists of fields and is principally a flat structure. Each field has a type and a name. However, by using previously defined message types, hierarchical structures can be constructed. Additionally, there is an "enum" construct that allows constructing enumerations. Fields of this type can have one value based on such a list. Instead of arrays, protocol buffers support "repeated" elements that support an arbitrary number of values of the given type.

The approach differs from C in two fundamental parts: each field must be assigned a unique numbered ID called tag. This tag is used when serializing data and is also used to enable backward compatibility. Second, it is possible to mark fields as required or optional. Optional fields are allowed to appear in serialized messages but can also be omitted. It is also possible to define default values for these fields. This capability is also a key element for ensuring forward and backward compatibility.

Protocol buffers support the following data types. Different encoding rules for the various integer types are the reason to have seemingly redundant types and are explained in the next section.

4.4.3.2.1 MODIFYING MESSAGE TYPES

One rare capability of protocol buffers is the possibility to change message types while still preserving backward and forward compatibility.

The most important rule is not to change the numeric tags. While the names of the fields are only used for the generated code, the tags are used for serialization. Second, additional fields should be optional or repeated. Although the serialization code ignores unknown fields and, thus, old implementation do not have to be changed if fields are added, the new code would report an error if a new required field is missing when receiving messages from an old system. The C++ and Java implementations even preserve unknown fields so that legacy systems can be on the path between current systems without interfering with the new message fields.

The language guide in the documentation for protocol buffers lists more rules for example even some type changes are allowed.

4.4.3.2.2 MODULARITY

Protocol buffers provide two functionalities for increasing modularity. First, it is possible to import the definitions of other .proto files to use the types defined there. Second, protocol buffers support namespaces in a similar way to Java: each proto file can define a package to prevent clashes when importing more than one file. These names are solely used to declare namespaces or packages when generating C++ and Java code respectively.

4.4.3.3 Encoding

One primary goal of protocol buffers is an efficient encoding. For this reason, some integer data types are encoded with a variable length. The most important primitive is the varint encoding.

4.4.3.3.1 VARINTS

The goal of varints is to allow smaller numbers to use less space and, thereby, save memory when the values are usually small. This applies for example also to the tag numbers that are usually very small integers. The encoding used by protocol buffers works on the byte level. The most significant bit (msb) of each byte indicates if the following byte has to be interpreted as part of the same number. The remaining 7 bits of each byte are

used for the actual value. The value bits of all connected bytes are concatenated (least significant bits are encoded first) and result in the actual value as shown in the following examples. The msb is shown in bold.

Value	Encoding (binary)	Concatenation/value (binary)
1	**0**000 0001	0000001
128	**1**000 0000 **0**000 0001	000001000000
300	**1**010 1100 **0**000 0010	00000100101100
1048575	**1**111 1111 **1**111 1111 **0**011 1111	01111111111111111111

Table 1: Protocol Buffers Varint Example

This encoding and decoding process can be implemented very efficiently by using shift operations and the omission of alignment and padding allows for a space efficient solution.

However, the previous encoding is not very efficient for negative values. Since CPUS represent negative values close to 0 as very large positive numbers (e.g., -1 =0xffffffff for 32-bit integers), encoding these values takes a lot of space. Therefore, protocol buffers also introduce the so-called ZigZag encoding. For this encoding, numbers close to 0 are represented as small positive values and encoded using the varint approach. The name ZigZag stems from the sequence of the encoding alternating between positive and negative numbers.

Again, this can be very efficiently done. A signed 32-bit value is converted to the encoding value with the following simple expression in C: `(n << 1) ^ (n >> 31)`

4.4.3.3.2 OTHER DATA TYPES

Besides the varint types, protocol buffers support also fixed-size integers with 32 or 64 bits stored in little-endian byte order. Additionally, floats and doubles are simply stored as 32 respectively 64 bits values as well.

The string type and the bytes type consist of a varint encoded length followed by the appropriate number of bytes.

Enums are encoded just by using the integer value and serializing it as a varint. Bools are also encoded as a varint of the values 1 or 0.

Therefore, there are the following types:

Encoding type	Type-ID	Used for
varint	0	int32, int64, uint32, uint64, sint32, sint64, bool, enum
64-bit	1	fixed64, sfixed64, double
Length-delimited	2	string, bytes, embedded messages, packed repeated fields
32-bit	5	fixed32, sfixed32, float

Table 2: Protocol Buffer Wire Types

The type IDs 3 and 4 are deprecated.

The different possible encodings are the reason for the availability of multiple integer types. The fixed32/64 and sfixed32/64 types are better used when the actual value range is big and also allow for a slightly speedier encoding. The difference between int32 and sint32 (and int64 and sint64 respectively) is the encoding of negative values: int32 are encoded as varints while sint32 values are encoded using the Zigzag varint type which is better optimized for negative values.

4.4.3.3.3 MESSAGE ENCODING

A message is encoded as a number of key-value pairs. The key of each field is the tag defined by the developer in the .proto file appended at the encoding type-ID (using 3 bits) with the following expression (in C): key = (tag << 3) | type_id

Therefore, a maximum number of 8 different encoding types are supported.

This key is then encoded as a varint followed by the value encoded in the appropriate format. Since the values of tags are usually small, most keys only require one byte.

A message consists of a number of key-value pairs. There is neither padding nor alignment. There is also no additional meta-data such as the overall size of the message, which instead must be provided when calling the deserialization function, or elements such as a CRC. If such meta-data is required, this must be handled by the application.

4.4.3.3.4 REPEATED FIELDS AND OPTIONAL FIELDS

Optional fields can be simply left out. If a message does not contain the key-value pair for an optional field, the field is indicated as empty. If a default value has been specified in the .proto file, this value is used by the deserializing function. Thus, the default value does not need to be transmitted over the wire.

There are two encodings supported for repeated fields. The older approach just allows multiple occurrences of the same key within a message. The values form an array based on the sequence in the encoding. An array of size 0 is simply omitted when serializing a message and, thus, does not use any space at all.

A newer solution allows for a more efficient encoding but requires explicit request by the developer in the .proto file by specifying the option `[packed=true]`. In this case, the key is only specified once followed by the varint-encoded number of bytes used by this field. The values are then encoded directly following each other using the respective type.

4.4.3.3.5 NESTED MESSAGES

Nested messages are encoded like strings: the tag of the field combined with the wire type length-delimited is used and the total number of bytes used for the content of the nested message is used as the length. The fields of the nested message (possibly containing nested messages as well) are serialized following the key.

4.4.4 Conclusions

While the REST ecosystem has evolved considerably in the last years, the maturity, prevalence and interoperability of SOAP based solutions are still far ahead. Therefore, we chose SOAP as the foundation for the platform.

However, depending on the results of the first prototype, we will consider the use of an alternative for the internal communication between components of the OESP.

Additionally, it would be possible to develop alternative APIs for the OESP based on other content encodings. However, while JSON has gained significant popularity for web application – not least due to its origin from JavaScript – the omission of a schema language and the limited gain in efficiency compared to XML when using compression, the use of a more efficient schema-supported binary format like Protocol Buffers or ASN.1 seems to be more suitable.

4.5 API Design for High Performance

Although most discussed solutions reduce the overhead incurred by the request-response mechanism, it is not possible to remove it completely. Therefore, as the most important approach to increase the performance, we designed the OESP APIs to inherently support batching operation. For example, it is possible to specify multiple metric types or attribute

types together with complex specifications of which entities are involved when requesting information from the EMS. This approach reduces the number of roundtrips between the clients and the OESP as well as between the OESP and the EMS and, therefore, both the delay and bandwidth requirements.

Second, the metric API allows the spatial and temporal aggregation of measurements to just transmit the necessary information to the clients.

Third, decoupling the applications from the data sources does not only simplify their implementations but also allows the platform to optimize the data flow among the various entities. This potential is further increased when using the publish/subscribe paradigm.

However, while this design decisions offers great potential to reduce the overhead, it must be actively exploited by the applications. It is therefore complementary to the approaches discussed before.

4.6 Conclusions

There are several potential starting points to improve the performance of web services. We are planning to enable the solutions to optimize the web service stack where implementations are widespread and the interoperability is not limited: SPDY and HTTP Content Compression. Additionally, depending on the experiences during the development and first evaluation phase as well as the evolution of the other discussed technologies, we will consider further optimizations at certain point in the system.

These optimizations for web services are complemented by the API design which allows applications to reduce the overhead considerably and enables the application to efficiently control the data flow between the entities involved in the system.

5 Conclusions

In this document, we present the preliminary specification and design for the Open Energy Service Platform. This document is closely related to and complements D2.1, which describes the overall architecture of the SmartKYE system and the interfaces provided by the OESP and by the energy management systems.

Starting from an analysis of the requirements gathered in WP1 and a discussion of important deployment considerations, we present the specification and design of the OESP. We discuss the decomposition of the OESP into individually deployable components to increase the scalability of the system. We also describe the communication mechanisms provided by the platform followed by considerations for the EMS descriptions and grouping service.

The second major part of the document discusses possibilities to increase the efficiency and performance of web services. We describe optimizations and enhancements for several layers of the web service stack.

The final version of the specification and design will be delivered in D3.3.

6 References and Acronyms

6.1 Acronyms

Acronyms List	
ABB	Architecture Building Block
AC	Alternating current
ADF	Architecture Development Framework
ADM	Architecture Development Method
API	Application Programming Interface
B2B	Business-to-Business
BAS	Building Automation Systems
BC	Business Cockpit
BIM	Building Information Modelling
BMS	Building Management System
BO&C	Building Optimization and Control
CEP	Complex Event Processing
COTS	Commercial Off-The-Shelf
DC	Direct current
DER	Distributed Energy Resources
DM	Dissemination Manager
DR	Demand Response
DSOs	Distribution System Operator
EC	European Commission
EGS	EMS of Generator System
EISP	Energy Information Service Provider
EMS	Energy Management Systems
EPL	EMS of Public Lighting
ESB	Enterprise Service Bus
ESCOs	Energy Service Company
ETL	Extract-Transform-Load
ETS	EMS of Traffic System
EU	European Union
EV	Electric vehicle
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HV	High voltage
HVAC	Heating ventilation and air conditioning
IaaS	Infrastructure as a Service

ICT	Information and communication technologies
ICT4EE	ICT for Energy Efficiency
IREEN	The ICT Roadmap for Energy-Efficient Neighbourhoods
IT	Information Technologies
J2EE	Java 2 Platform Enterprise Edition
JMS	Java Message Service
KPI	Key Performance Indicator
MCC	Monitoring and Control Cockpit
MMI	Man Machine Interface
MUN	Municipality
MV	Medium voltage
NFRs	Non-Functional Requirements
SmartKYE	Smart grid Key Neighbourhood Indicator Cockpit
OASIS	Organization for the Advancement of Structured Information Stand-ards
OESP	Open Energy Service Platform
OMG	Object Management Group
PaaS	Platform as a Service
PHEV	Plug-in Hybrid Electrical Vehicles
PLS	Public lighting system
PPP	Public-Private Partnership
PV	Photovoltaic
QoS	Quality of Service
RA	Reference Architecture
RES	Renewable Sources
REST	Representational State Transfer
SaaS	Software as a Service
SCADA	Supervisory Control and Data Acquisition System
SLA	Service-Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TOGAF	The Open Group Architecture Framework
TS	Time Series
TS Data	Time Series Data
UC	Use Case
URL	Uniform resource locator
WP	Work packages
WSDL	Web Services Description Language

6.2 References

Aziz, A., & Mitchell, S. (2007). *An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET*. From http://msdn.microsoft.com/en-us/library/bb299886.aspx

BSON. (n.d.). From http://bsonspec.org/

Google. (n.d.). *Protocol Buffers*. From http://code.google.com/apis/protocolbuffers/

Internet Engineering Task Force (IETF). (2012). The OAuth 2.0 Authorization Framework.

ITU. (n.d.). *Introduction to ASN.1*. From http://www.itu.int/ITU-T/asn1/introduction/index.htm

Jäger, K. (2007). From Introducing BISON - Binary Interchange Standard and Object Notation : http://kaijaeger.com/articles/introducing-bison-binary-interchange-standard.html

JSON. (n.d.). From http://www.json.org/

Microsoft. (n.d.). *Microsoft Speed + Mobility* . Retrieved 10 15, 2013, from http://tools.ietf.org/html/draft-montenegro-httpbis-speed-mobility-02

Roskind, J. (2013, 06 24). *QUIC: Design Document and Specification Rational*. Retrieved 10 15, 2013, from https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit

The ASN.1 Consortium, Inc. (2003). *ASN.1 developments*. From http://www.asn1.org/news.htm

The Chromium Projects. (n.d.). *SPDY: An experimental protocol for a faster web*. Retrieved 10 15, 2013, from http://www.chromium.org/spdy/spdy-whitepaper