Architecture Committee Handbook

openKONSEQUENZ

created by

Architecture Committee

We acknowledge that this document uses material from the arc 42 architecture template, http://www.arc42.de. Created by Dr. Peter Hruschka & Dr. Gernot Starke.
Template Revision: 6.1 EN
June 2012

Revision History

Version	Date	Reviser	Description	Status
1.0	2016-07-04	A. Göring	Alignment in AC/QC conference call	Released
1.0.1	2016-07-19	A. Göring	Added UML-Tool decision in chapter 2. Constraints, Added software-tiers image in chapter 8.	Draft for v1.1
1.1	2016-08-18	A.Göring	Alignment in AC/QC conference call	Released
1.1.1	2016-08-26	F. Korb, M. Rohr	Description of architecture layer model and its APIs. Example internal module architecture (Presented in ACQC-Meeting 15.& 29.08.2016)	Draft for v1.2
1.2	2016-09-14	A. Göring	Integration of Concept for Plattform Module Developmennt, Consolidation v1.1.1	Released
1.2.1	2016-09-16	S.Grüttner	Reorganization of Chapter7 Deployment Environment, clearifying the reference environment as “image”. Adding cutting of CIM Cache. Modified Logging (8.17) for use of SLF4J. Added potential non-functional requirement for Offline-Mode.	Draft for v1.3
1.2.2	2017-01-30	A. Göring	Adding Link to oK-API Swagger Definition, deleting old Interfaces Annex. Adding CIM Cache Module dependencies image and text (from Felix Korb)	Draft for v1.3
1.3	2017-02-14	A. Göring	Alignment in/after AC/QC conference call	Released
1.3.1	2017-09-05	A. Göring	Minimum requirement change from Java EE 7 to Oracle Java SE 8. Added reporting tool decision as discussed in AC/QC conference call 2017-08-28	Released

Formal

According to a decision of the Quality Committee, this document is written in english.

Document control:

Author: Andre Göring, andre.goering@offis.de (assistant of architecture committee)

Reviewed by: SC, PPC, and QC of openKONSEQUENZ

Released by: AC

Released versions will be made available via the openKONSEQUENZ web site.

Open Issues for Architecture Committee Handbook

This is a living document. Further general architectural topics have to be detailed by the Architecture Committee and can not yet be answered without further knowledge from further openKONSEQUENZ projects. Known issues are listed red coloured.

Module specific Architecture Documentation Hints

GREEN & Boxed: Open architecture documentation issues for module developers in **module specific** arc42-document or tender or global oK-CIM-Profile.

Related documents

Document	Description
BDEW Whitepaper	Whitepaper on requirements for secure control and telecommunication systems by the german BDEW Bundesverband der Energie und Wasserwirtschaft e.V. (https://www.bdew.de/internet.nsf/id/232E01B4E0C52139C1257A5D00429968/$file/OE-BDEW-Whitepaper_Secure_Systems%20V1.1%202015.pdf)
BSI TR-02102	Technical Guideline according to encryption recommendations and key length by the german BSI - Bundesamt für Sicherheit in der Informationstechnik (https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr02102/index_htm.html)
oK-API	The APIs to connect modules to each other as well as source systems such as DMS, GIS, ERP to platform components are defined with Swagger (http://wiki.openkonsequenz.de/Source_System_API)
oK-Charter	The openKONSEQUENZ charter (https://wiki.eclipse.org/images/f/f5/20150623a_openKonsequenz_V14-3_%283%29.pdf)
oK-GUI-Styleguide	Style guide for module developers of openKONSEQUENZ modules according to the graphical user interface. (http://wiki.openkonsequenz.de)
oK-Module-Tender-Call	The openKONSEQUENZ project planning committee prepares a document which describes the requirements to the development for each module. With this document it calls for tenders at software developers (module individual)
oK-Module-Tender	The software developers answer to the oK-Module-Tender-Call (module & developer individual)
oK-Vision	The oK document “Vision/Mission/Roadmap” - it is currently not available online.
oK-Website	The website of openKONSEQUENZ (www.openkonsequenz.de)
Quality Committee	Textural guideline for module developers of openKONSEQUENZ

Handbook	modules according to quality related issues. (https://wiki.eclipse.org/OpenKONSEQUENZACQCRichtlinien)

Table of Contents

1. Introduction and Goals

The consortium openKONSEQUENZ (oK) consists of Distribution System Operators (DSO), their software manufacturers, integration service providers and academic institutes. It targets to overcome vendor lock-in in the hierarchical grown IT-infrastructure of DSOs. The openKONSEQUENZ® Working group (openKONSEQUENZ® WG) wants to foster and support an open and innovative eco-system providing tools and systems, qualification kits and adapters for standardized and vendor independent e-Systems. Therefore openKONSEQUENZ defines a reference platform, where different modules for different purposes shall be implemented for and interact with existing parts of the IT landscape and other modules under development.

This handbook describes the architectural view on openKONSEQUENZ software, its current overall system and software architecture which developers shall develop in respect to and what software architecture artifacts (documentation & graphs) developers have to document during development. In especially the “green colored TODOs” mark important documentation that module developers must create.

These artifacts have to be stored in Arc42 - architecture documentation for each module in the module specific git-repository (see Quality Committee (QC) handbook for naming and directory conventions). The architecture guidelines are generally applicable. These guidelines can be adapted over time based on specific need and/or request from Architecture Committee (AC) members or third party application developers. A change management process will be set up for this in the future.

1.1 Requirements Overview

The openKONSEQUENZ is an umbrella for a rising number of projects with common intends:

Functional Requirements:

- Provide a common interoperable platform, which can unite the data of different existing systems and offers a streamlined environment for extending these existing systems by new User Modules. The oK uses open interfaces and reduces or optimizes interfaces where applicable.
    - What happens: different systems share data along the platform.
    - Why:
        - Islands in IT landscape shall be connected in an easy way
        - To overcome vendor dependent interface development and vendor lock-in.
- Get software support for new and demanding requirements in context of the energy policy turnaround.
    - What happens: New modules realize functions on existing information
    - Why: New functions required to easily solve problems / processes.

Non-functional Requirements:

- Requirements for modules according to confidentiality, availability and integrity are defined in levels normal, high and very high (normal is descripted in BDEW Whitepaper). The oK reference platform has to be designed for meeting specification for levels high to very high.
- Detailed and hard requirements for the oK reference platform shall be listed in Section 10 Quality Scenarios.

Different modules may not only underlie different functional requirements, but also different non-functional requirements. These are specificated in the tenders for each module. So the entirety of requirements for modules comes from tender documents, the AC and QC handbooks, SCRUM product backlog and SCRUM sprint backlog.

1.2 Quality Goals

Overall Quality Goals of openKONSEQUENZ are detailed in the openKONSEQUENZ Charter and listed in short:

- process and data integrity with standardized interfaces,
- long term maintainability for components to be usable longer than 15 years,
- compliance with frequently changing regulation,
- vendor-neutrality,
- availability as needed,
- security for critical infrastructure by design,
- innovations in products and development.

There is a need to make a difference between openKONSEQUENZ User Modules, Platform Modules (Domain Modules and Core Modules) and a reference platform itself to qualify goals. The reference platform is a standardized host for new modules. Platform Modules are modules that are to be used in several projects for supply of data or services. User Modules are that applications, a user from a DSO uses for solving their use case. (For detailed descriptions see chapter “Solution Strategy”.) For each part, the Quality Goals have to be discussed individually. Please check also Quality Committee Handbook for quality related requirements.

Reference Platform and Platform Modules

- Flexibility - The reference platform shall allow, that different systems and modules from different vendors/developers can interact and interoperate, and may be exchanged or recombined.
- Availability - All platform modules that are running on the platform can only be as available as the platform - same for user modules that are based on platform modules.

- Maintainability (and testability as part of maintainability) - platform and its platform modules shall be used longer than 15 years.
- Integration performance - new implemented functionality of oK own modules and external modules shall be included fast / automatically.
- Security - the platform and its modules need to underly security-by-design

User Modules:

- Functionality - user modules must fulfil their functional requirements
- Integration performance - user modules must be easy integratable in different production environments.
- Modifiability (and testability as part of modifiability) - Good documentation (i.e. code and architecture documentation) makes code changes easier and automatic tests facilitate rigorous verification.
- Ergonomics - according to oK-GUI-Styleguide.

1.3 Stakeholders

DSOs

Need software fulfilling their functional and non-functional requirements.

AC

Manages openKONSEQUENZ Architecture demands and is responsible for this document.

QC

Manages openKONSEQUENZ Quality demands and is responsible for the related Quality Committee Handbook.

Module Developer

Is the software developer who develops a module (or the tender for the module). He has to take the QC handbook and this AC handbook into account, when offering a tender and when developing a module.

System-Integrator

Needs information for integration of modules in a specific DSO environment.

TODO for architecture documentation
by module developer according to chapter 1:

- Document defined module specific functional and non-functional Requirements/Quality Goals/Acceptance Criteria in the module project's git-repository in an arc42-document.

2. Architecture Constraints

In the following section, architecture constraints are listed. This are any requirement that limit software architects in their freedom of design decisions or the development process:

- License: Open Source-Licence “Eclipse Public License 1.0” with its demands on Intellectual Property (IP) for the usage of libraries.
- Commissioning/Hiring/Business: In the consortium, one or more of the DSOs is/are the driving force for a module. In procurement the Lead Buyer (N-ERGIE AG) is the contactpoint for business questions. In contrast to this the driving DSO is handling functional/technical questions and is responsible for commissioning.
- Development using SCRUM.
    - Product Owner comes from the specific module driving oK DSO
    - SCRUM Master from the module developing company.
- Quality Control and Acceptance: Sprints + 3 month test operation + See QC handbook.
- Standardization: Usage of standardized data structures (CIM) and the reference platform.

An additional architecture constraint is defined in the chapter “Solution Strategy”, which describes the oK Multilayer Architecture.

2.1 Technical Constraints

Technical frameworks or requirements constraining the developing of modules are largely based on the experience made with the initial pilot application implementation. These are defined to allow interoperability between the various modules to come and the oK reference platform and for oK reference platform enhancements.

When an application developer would like to use different components than those listed below, he has to get approval from the Architecture Committee.

Software platform framework and requirements	
Basis components of the reference platform	Portal Liferay; Application Server Tomcat; JPA EclipseLink; Database PostgreSQL; ESB Talend (open studio for ESB) BPMN Engine Camunda
Runtime engine	Java Oracle 8 SE

ESB-Interfaces	openKONSEQUENZ-CIM-Profiles for APIs based on CIM 17 or later via CIM RDF/XML for topologies or deep nested structures and XML for others via RESTful Webservices.
GUI	See oK-GUI-Styleguide
Programming Language GUI	AngularJS, Bootstrap, jQuery, REST/JSON Interfaces
Libraries, Frameworks, Components	Used Libraries/Frameworks have to be compatible to be used in an Eclipse Public License project. Maven, For Libraries for quality check and Continuous Build/Deployment/Integration see QC Handbook
Programming Constraints	
	See QC Handbook
UML-Tooling	For CIM related modeling, the use of the tool Sparx "Enterprise Architect" (EA) is strongly encouraged. If other tools are used, a data transfer (export/import) to EA must be frictionless, i.e. model structure, contents, and diagrams have to be transferred to a EA model (possible). For software architecture related modeling, an open source UML tool shall be used. Up until further notice, this tool is "Modelio".

This framework and set of products describes the openKonsequenz reference system components. Instantiations on grid operator site may get adapted on request by the grid operator or on recommendation of the system integrator to cope with specific grid operator requirements regarding i.e. integration with existing ESB, BPM or application server technology.

2.2 Organizational Constraints

For architecture constraints responsible persons/groups are listed below.

Organization and Structure	
Steering Committee (SC)	Has the final say in every oK question.

Project Planning Committee (PPC)	Plans following projects and specifies requirements for each individual module. Calls for tenders based on their requirements, the AC handbook, QC handbook and oK-GUI-Styleguide.
Architecture Committee (AC)	Gives Framework and Constraints for architecture according to technology, interfaces, reference environments, architectural concepts, design decision and documentation. Every difference from their architecture guidelines in module development has to be approved and agreed by the AC. The AC may in case of deviation advise the Product Owner to refuse or accept a sprint acceptance or product acceptance in architectural questions.
Quality Committee (QC)	Gives Framework and Constraints for quality of the software, like code quality & styleguides, testing, build. These quality aspects also mirror back to architecture questions (and vice versa).
Product Owner (PO)	Gives and prioritizes requirements and acceptance criteria in Product Backlog for Sprint Planning / Backlog and checks their fulfilment in sprint reviews and may by this limit architectures solution space of a module. The PO may refuse a sprint or product acceptance cause of non-fulfilment of architecture requirements.
SCRUM Master (SM)	Is responsible for the success of the SCRUM. SM checks for Compliance with the SCRUM Process and ensures as well the communication between module developers and PO. SM helps PO at maintaining the Product Backlog and the developer at the definition of done.

The modules are developed in Eclipse Projects. User modules may be developed in closed projects, where only the contractor for module development is a Committer (According to the committer rules of Eclipse Project). Platform Modules (Domain Modules and Core Modules) shall be developed (further) in the oK platform project, where oK-Core-Developer (that need to be established) are Committer.

2.3 Conventions

- Architecture related module specific documentation has to be documented in the module specific git repository (see QC Handbook for directory & type advises) according to the Arc42 template. Documentation language is english. In especially

the “green colored TODOs” mark important documents/arc42-parts, that have to be created by module developers. Use UML for graphs! (For Tooling see chapter 2.1).

- See “QC Handbook” (for coding styleguides, quality reports, naming conventions, version and configuration management...)

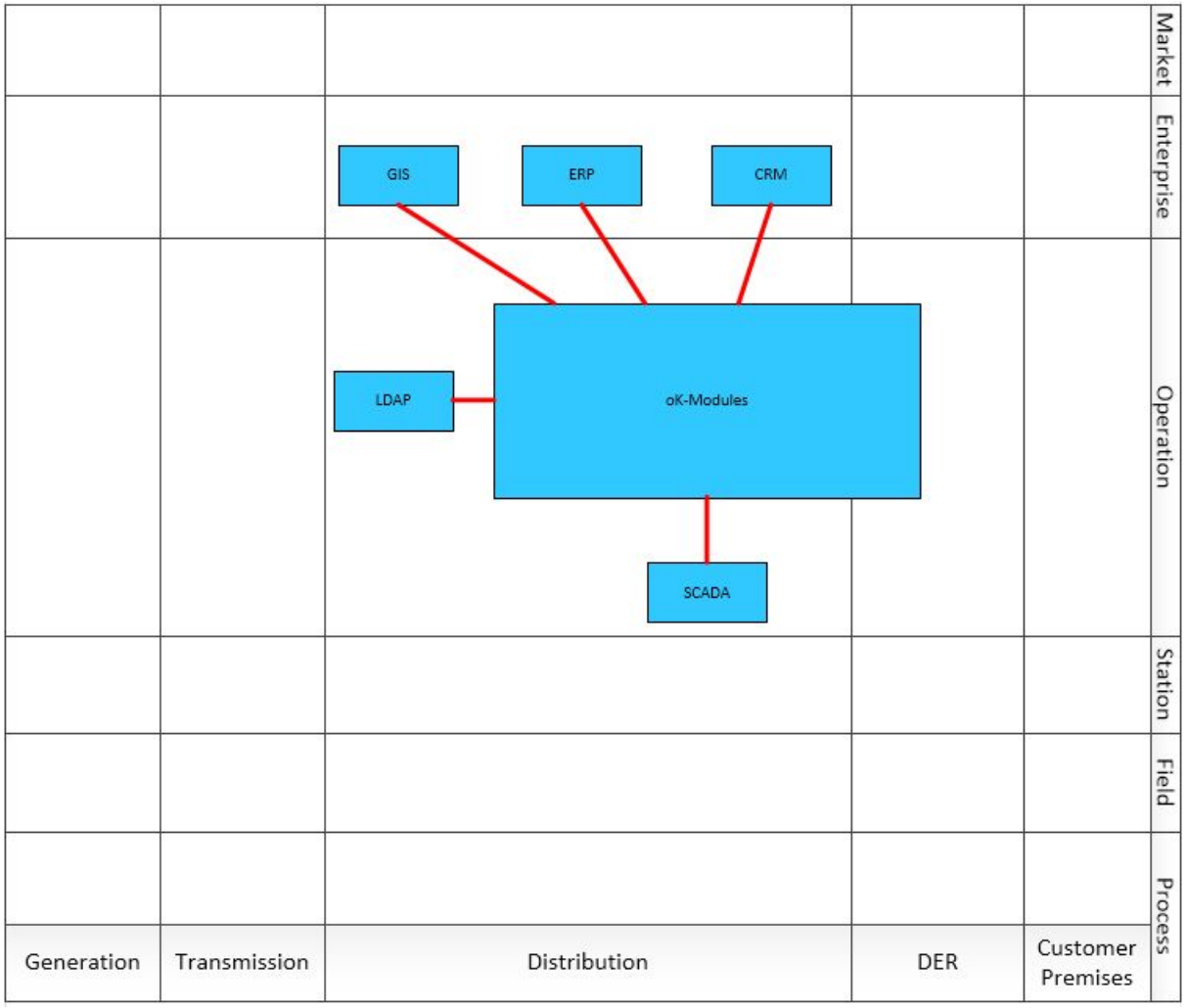
3. System Scope and Context

The openKONSEQUENZ develops modules for the operation of distribution grids according to current IT technologies and regulation. These modules extend the functionality of SCADA, often coupled with other business systems of a DSO. The openKONSEQUENZ hosts User Modules and Platform Modules. The User Modules offer extended functionality, that the current solutions do not provide (either in functionality or in quality). The Platform Modules (Core Modules and Domain Modules) provide domain-independent and energy domain-applied services, that are needed for a variety of User Modules, external modules or even the current existing IT systems. Therefore the modules communicate with each other and with the existing IT infrastructure via interfaces to gather and distribute the necessary data for operation.

- Main focus is the communication between different modules on business logic tier to ensure communication between different systems in different settings from different DSOs.
- As transport medium, the Enterprise Service Bus is required to handle all communication between User Modules and Platform Modules as well as to existing business systems.
- No direct communication between modules and especially no communication via direct access to the same database schemes shall exist.
- Also the tier-wise communication between graphical user interface and business logic is in focus of openKONSEQUENZ.

3.1 Business Context

In openKONSEQUENZ additional modules for distribution network operation are developed, that are usually located outside of a SCADA system. A long term view is shown in the following picture according to the Smart Grid Architecture Model plane. A use of openKONSEQUENZ modules is also imaginable in the TSO domain, the components shown in the figure then move one column to the left. Module developers have to have in mind, that DSOs often separate the operation zone into common operation and SCADA.



Neighbouring systems of modules are

- existing DSO systems
  - SCADA,
  - GIS,
  - ERP,
  - CRM,
  - Weather forecasts
  - ...
- Other Platform Modules
  - Topology Management(pilot)
  - Archive (planned)
  - Identity & Access Management (planned)
  - ...
- Other user modules
  - Eisman (feed-in management; pilot)
  - Operation Diary (planned)
  - Switch Request Management (planned)
  - ...

3.2 Technical Context

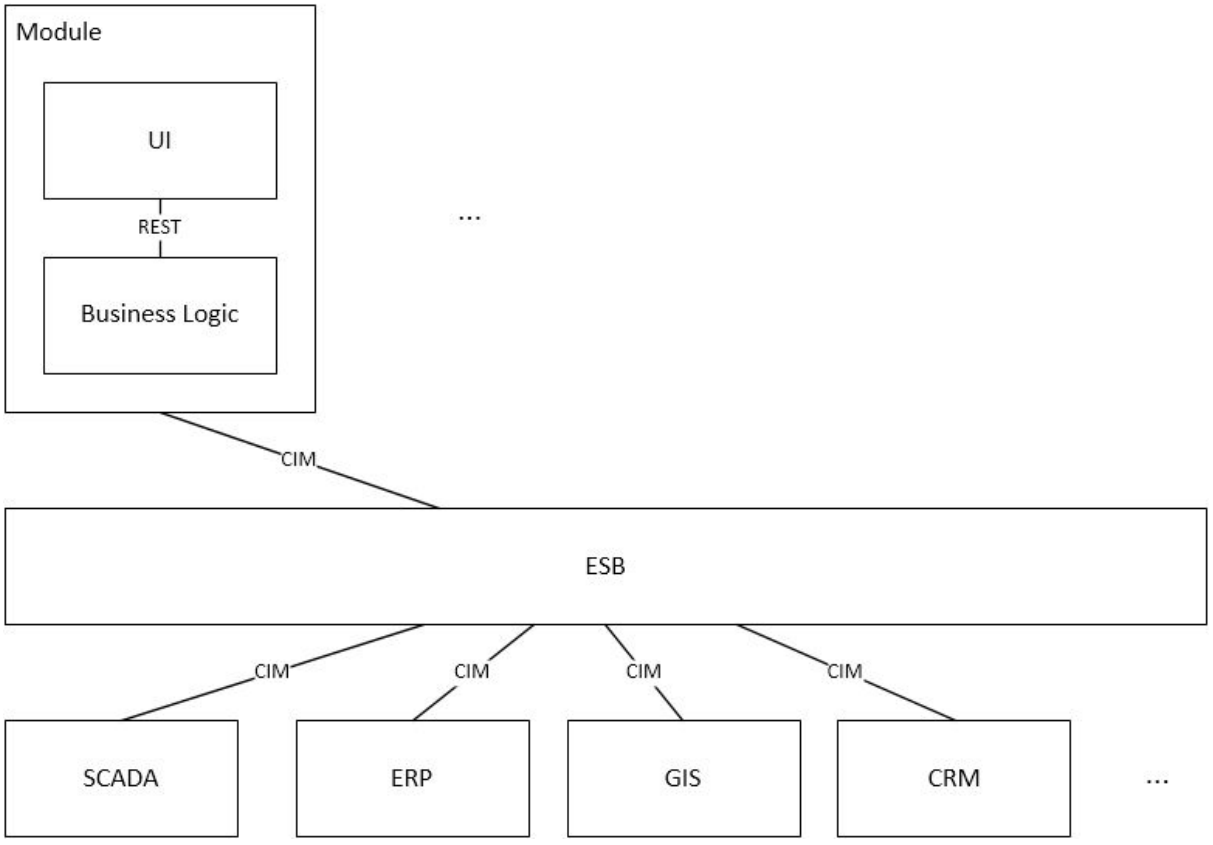
The typical system landscape of DSOs is characterized by heterogeneous data models in closed legacy systems with proprietary interfaces. Because of this, there are high integration barriers and -costs that slow down grid operators and software suppliers (redundant data management, inconsistencies, adapter dev.,...). The oK approach is to drive open interfaces/data models based on CIM standard with standardized semantics and less misunderstandings and proprietary knowledge.

According to the following figure, there is to differ between inner module communication between the business layer and the UI layer (called REST in the figure) of a module and module external communication or inter-module communication (called CIM in figure).
For the module external communication (called CIM in figure) interfaces to all systems of DSOs have to be taken into account under following aspects:

- As a basis, the Common Information Model - CIM (in current development Version 17 or newer) is/has to be used for definition of interfaces. It provides an ontology for equipment in the electrical domain, giving semantics and syntax of attributes, associations and classes in an object oriented way.
- This includes the use of serialization format RDF (CIM/XML) for topology or deep nested structures and XML for all others as well as the use of CIM-envelopes (See IEC 61968-100).
- If a module is not using this serialization format, it has to be reasonably explained and documented and needs an authorization of the ACQC.
- As interface-technology RESTful web services are used.
- For each external interface (interfaces between modules or external systems) the interface has to be documented in this document.
- Interfaces have to be designed under the interface segregation principle.
- For checking out interoperability of interfaces, the existing overall openKONSEQUENZ interface schema must be extended by the module developer - this shall ensure reutilization of existing parts of the schema and avoid inconsistencies in semantics/syntax.
- Dependencies of modules to services realized by other modules have to be specified and documented explicitly.
- When CIM is not appropriate (like access management), other standards in their respective domain shall be taken into account first to avoid proprietary and inaccurate interfaces. The interface has also be documented in the overall openKONSEQUENZ interface profile and it should use REST & XML - otherwise reasonably explained and documented exceptions have to be approved by ACQC.
- In the future a development of shared DSO CIM profiles comparable to entso-e profiles is planned: The particular models of a certain grid relating messages or files

are conform to the overall oK-DSO-CIM-profile specification, which itself shall be conform to the official (future) CIM standard.

For inner module communication (called REST in the figure) RESTful Webservices shall be used with json serialization, which fits well for the JS-GUI Tier. This is common to User Modules and Platform Modules. If the latter does not need an UI, the REST called interface (in the figure) and UI can be left away.



3.3 External Interfaces

Current overall oK interface profile

The overall oK interface profile is hosted in the git-Repository of openK-platform (openk-platform/openk-cim-v17) as Java packages and classes with Java documentation for the semantical use of attributes/classes and as Image of an UML class diagram. If extensions are needed for modules, the module developer has to update this overall model with needed attributes/classes and their semantical meanings. Thereby it shall be granted, that module developer take the current available model as basis, that no redundant information is in the profile and that there is no semantical inconsistency introduced.

Short interface description for each oK-ESB-interface

For each ok-ESB-interface, a module provides/requires, a detailed description of the interface can be found in the module-specific Arc42-documentation. An overview, which

introduces all ok-ESB-interfaces and lists provider and requirer, can be found in the oK-API document.

TODO for tender
by module developer according to chapter 3:

- Specify provides- and requires-interfaces for the planned module as a first draft (if there are no yet fitting oK-interfaces).

TODO for architecture documentation
by module developer according to chapter 3:

- Define module specific scope and context in the module own architecture documentation (in projects git repository)
- Get overall oK interface profile from ok-platform git repository, update it, if current version of oK interface profile lacks classes/attributes and commit changes to ok-platform git repository. Eventually update the oK-API document.
- Describe own external interfaces (requires and provides) in architecture documentation, create their UML models and XML Schema Files (in respective git-repository-directory (see QC-Handbook for directory details). Update the oK-API document.

4. Solution Strategy

In the tenders for a module, the module developer has to provide an overall solution architecture and to define which oK modules are planned to be reused by the module. The module developer has to specify changes to existing modules, if required. Furthermore, it is required to follow the oK Multilayer Architecture that is described in this chapter. In the case of deviation from the oK Multilayer Architecture, a module developer has to ask the ACQC committee to come to an agreement.

Existing systems, (possible) externally developed modules, oK User Modules, and oK Platform Modules (Core Modules and Domain Modules) interact on basis of standardized interfaces (the oK APIs) and run on a reference architecture concept underlying system. A portal and a UI-Styleguide make it easy for users to work with new modules.

In the tender, the developer lists the libraries that will be used (an IP-check on each new library is required and shall be the responsibility of the module developer). If libraries with the necessary capabilities are already listed as “cleared”, they shall be used as a default; any project who wants to override the default list needs to present rationale and needs the approval of the architecture committee.

IP-checks can be long-running tasks. It is strongly recommended to perform an IP-check before creating the tender. Any planning, estimation, design or implementation based on a library that does not pass the IP-check is wasted.

The remainder of this chapter describes the oK Multilayer Architecture in general before the elements of the architecture are described in detail.

TODO for module developer in tender:

- provide overall solution architecture that fits to the oK Multilayer Architecture, list of reused modules, changes to reused modules, list libraries that will be used.

TODO for module developer according to Arc42-documentation:

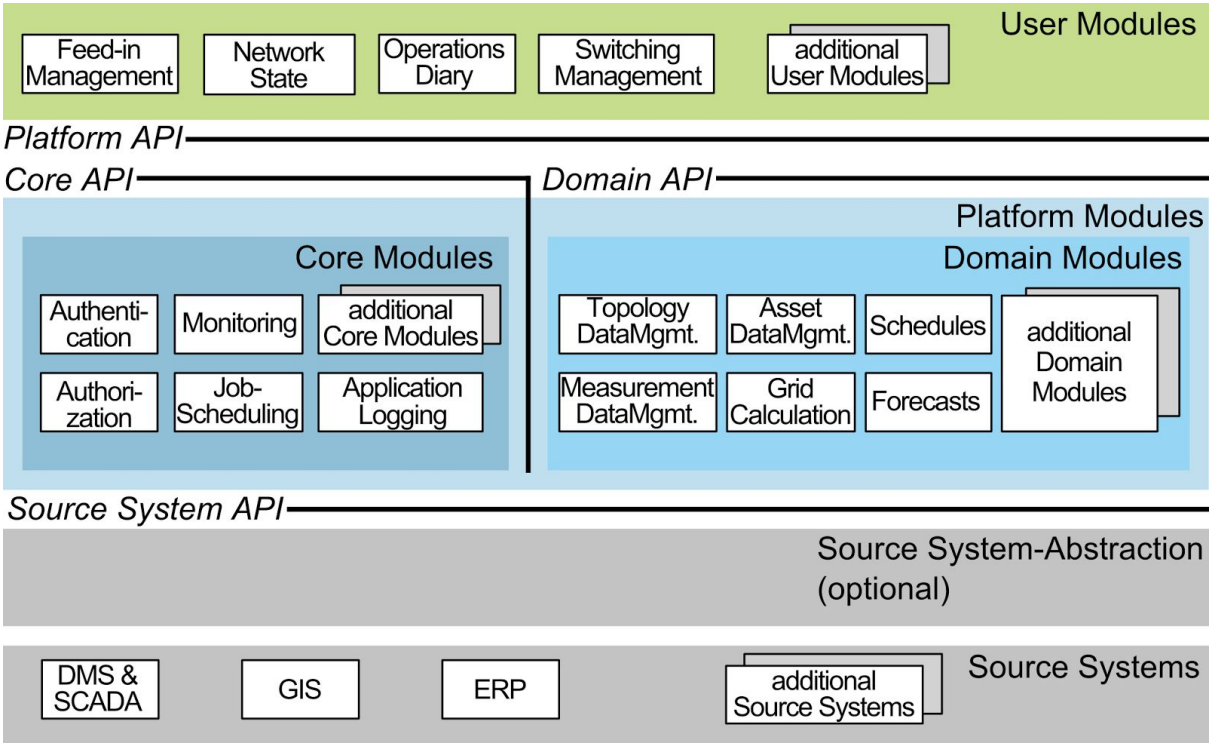
- Describe solution strategy for each module in module own architecture documentation

4.1 oK Multilayer Architecture

The following illustration shows the oK Multilayer Architecture. It contains four layers and APIs in between. This architecture provides a general structure for the openKONSEQUENZ software. Each oK module has to be located at some point in this architecture and the communication of the modules has to use the corresponding APIs. The goal of this architecture is to support reusability and understandability for users, module developers, and system integrators.

The four layers (User Modules, Platform Modules, Source System-Abstraction, and Source Systems) contain components, systems and adapters. The Platform Module layer distinguishes Core Modules and Domain Modules.

The APIs in the oK Multilayer Architecture represent a set of concrete APIs between the layers. The APIs are an important part of the vision of openKONSEQUENZ in terms of standardization and modularization. The Platform API is between the User Modules and the lower layers and the Source System APIs separate the openKONSEQUENZ software from the Source Systems and the Source System-Abstractions. The Platform API itself combines the Core API and the Domain API.



4.2 Platform Modules

The Platform Modules Layer in the oK Multilayer Architecture consists of the Platform Modules (containing the Core Modules and the Domain Modules).

Currently, openKONSEQUENZ has a pilot module that manages an integrated topology model and there are ongoing discussions how to systematically develop the platform. The initial eclipse project oK-Platform currently hosts the platform but also temporarily contains an User Module.

4.3 Platform API

The Platform API is the set of all interfaces (and versions) for the oK platform modules. It is expected to be used by User modules.

4.4 Source Systems

In the Source Systems layer are the typical systems of a DSO, such as:

- GIS (geographic information system)
- SCADA (supervisory control and data acquisition)
- DMS (distribution management system)
- ERP (enterprise resource planning)

These Source Systems are usually proprietary solutions with mostly vendor-specific proprietary interfaces. In some cases, even solutions from the same vendor have a customer-specific configuration and customer-specific data structures. The source systems in the proper sense are not part of the openKONSEQUENZ solution.

These Source Systems are responsible for a large part of the data that is used by the oK modules. These systems are responsible (leading; “führend”) for a certain subset of the data that is relevant for the operational use cases of a DSO. For instance the DMS provides information about the actual grid status, the GIS provides detailed information about the static topology, and the ERP provides master data for instance for the renewables.

4.5 Source System-Abstraction

Usually, it is not possible to directly connect the Source Systems with the oK Platform. The Source Systems usually only have interfaces that are proprietary, low level, or without specified machine-readable semantics because of the lack of adequate standards. The “Source System-Abstraction” layer contains adapters that implement the Source System API. It is expected that adapters in this layer will have to do more than just value mapping and therefore requires explicit programming.

This layer can be optional in the case when a future release of a source system directly supports a particular oK API.

4.6 Source System API

The Source System API provides interfaces between source systems and the oK modules. This API defines a common view on Source Systems for the oK modules. This simplifies the development of oK modules as only the API has to be addressed - this API abstracts from details of the Source Systems of a DSO. Additionally, the Source System API generalizes from vendor specific proprietary interfaces to allow the development of User Modules that are independent from Source System vendors. Moreover, the Source System API decouples the User Modules from the Source Systems, so that changes in the source system landscape of a DSO has far less impact on the User Modules than without such an API. Technically, this API could be considered an application of the facade pattern.

4.7 Domain Modules

The Domain Modules provide domain-specific services to multiple User Modules that go beyond the services that are provided by the source systems (via the Source System API). The main idea behind Domain Modules is to have focused User Modules with little overlap and Domain Modules that provide shared services for User Modules. See also the design decision “Service Components for more than one Module” in the design decision chapter.

Domain Modules can provide a optimized or even required way to access Source System data for instance by integrating the data of multiple Source Systems and by caching Source System data. The caching can be suitable to increase the data availability from Source Systems that are not high-availability and the caching can prevent that User Modules cause too much load on the Source Systems.

For a concrete installation of openKONSEQUENZ software at a DSO, it is only required to install the subset of the Domain Modules that is required by the user modules the DSO wants to use. Therefore, a DSO only needs to implement adapters for a subset of the oK Source System API.

Multiple implementations of the same Domain Module can exist, and a Domain Module implementation might be only a wrapper for a non-oK module. For instance, it can be beneficial to have multiple implementations for providing renewable feed-in forecasts even in the same system.

openKONSEQUENZ aims to provide for each Domain Module an own open source implementation or an adapter to an open source library which provides the required functionality.

Topology management

The topology management (formerly called CIM-Cache) is a Domain Module that hosts an integrated topology data module and services to access it. The data is integrated from multiple Source Systems. In the future, Source Systems shall automatically update relevant changes.
So far, a pilot implementation of the topology management exists which focuses on feed-in-management ("Einspeisemanagement"). Currently, (08/2016) an API is specified for the topology management.

4.8 Domain Module API

The Domain Module API is the set of all interfaces (and versions) for the oK Domain Modules. The Domain Module API is used by User Modules, by Domain Modules, and also by Source Systems (e.g., for publishing updates). Core Modules must not depend on the Domain Module API.

4.9 Core Modules

Core Modules provide services for cross cutting concerns (https://en.wikipedia.org/wiki/Cross-cutting_concern) in a standardized way for the oK-modules. In contrast to Domain Modules, Core Modules are not specific to the domain of openKONSEQUENZ (which is described in the system context chapter; i.e. software for operating distribution grids).

It is expected that openKONSEQUENZ does not need to provide own implementations for most Core Modules because suitable open source libraries exist elsewhere. For those cases, the Core Modules will only be adapters that implement the Core API to provide an oK standardized way to connect the external open source library.

Candidates for Core Modules are:

- Access Control (Authentication and Authorization). There is no implementation of this module in oK, so far. It could be implemented using for instance with the open source solution Keycloak.
- Common logging
- Monitoring: This module is responsible for technical monitoring in terms of verifying that the oK services are operational and alerting if this is not the case. There is no implementation of this module in oK, so far. It could be implemented using for instance with the open source solution Icinga. This monitoring should not be

confused with the User Module “Betriebstagebuch” (Operation Diary): The technical monitoring is primarily intended for IT staff while the Operation Diary is used by the grid operation staff of the DSO.

- Reporting

4.10 Core Modules API

The Core Modules API is the sum of all interfaces (and versions) for the oK Core Modules. The Core Module API is used by Domain Modules and by User Modules (via the Platform API).

4.11 User Modules

The oK User Modules are the domain applications in openKONSEQUENZ. Typically they implement one or multiple related use cases in the domain of openKONSEQUENZ (see system context chapter) and have directly interaction with end users via user interfaces. They only access data and services via interfaces that are defined in oK. The user modules can use all oK APIs (Source System API, Platform APIs (Core Module API, Domain API)). However, the User Modules have to use oK Platform APIs if these provide required data from source systems instead of directly accessing the source systems via the Source System APIs. There can be exceptions to this constraint.

5. Building Block View

The openKONSEQUENZ building blocks are the User Modules and the Platform Modules (Domain Modules and Core Modules), as specified in the oK Multilayer Architecture in the chapter on the solution strategy. All oK Modules are independent software components. They’re connected to each other by provides and requires interfaces. New modules can be build using existing modules interfaces and information (via the interfaces).

5.1 Level 1

TODO for architecture documentation
by module developer according Building Block View Level 1:

- Document, which modules are reused with which interfaces as UML component diagram in the oK-API document.

Core API

Technical services (e.g. authentication/authorization) are accessible via the core API. The technical services shall be used by all domain modules and all user modules, their functionality is offered centralized by the oK platform. The oK architecture will evolve over time offering more and more core modules. The repository of technical services will be built up incrementally with each project. Whenever a project needs such a technical service, it has to specify its requirements to the functionality and API of the service.

There are three possible scenarios:

1. The oK service exists, and the API and functionality of the oK service are sufficient for the project's needs.
   The project documents its usage of the service. The service is modeled as an external actor of the projects domain or user services.
2. The oK service exists, but adaptations are required to meet the new project's requirements.
   The project has to document a change request based on its requirements to the service. The AC/QC will review the required changes. If the AC/QC releases the change request, the change will be implemented (typically by the project itself). If the AC/QC rejects the change request, the project has to use the API and functionality as it is.
3. An oK service does not yet exist, it has to be defined as new technical service.
   The project will modify/extend its requirements to take into account future re-use of the service. Based on these generic requirements, the project suggests functionality and API for the new technical service. The AC/QC and the PPC will review the new definition. If the definition is rejected the, the project has to re-iterate the specification and the new definition will be reviewed. If the definition is accepted, the project shall implement a prototype for the service (Ideally, the project uses an open-source component that fulfills the requirements and has been cleared by an IP management process). The implementation shall be managed in its own project structure to be independent from the projects domain functionality.

A project shall declare its usage of core APIs at bidding time, the tender has to contain a list of technical services required. For each technical service, the project shall document the usage scenario according to the list stated above.

For example, a project may specify that it needs an authentication service. The project reviews the core API of openK and finds the authentication service to be sufficient. The project documents this by indicating Scenario 1 for authentication. On the other hand, a project may specify "Technical Service used: Monitoring module health – Scenario 3" which means that the project wants its domain services monitored, but the API required to do this doesn't exist yet.

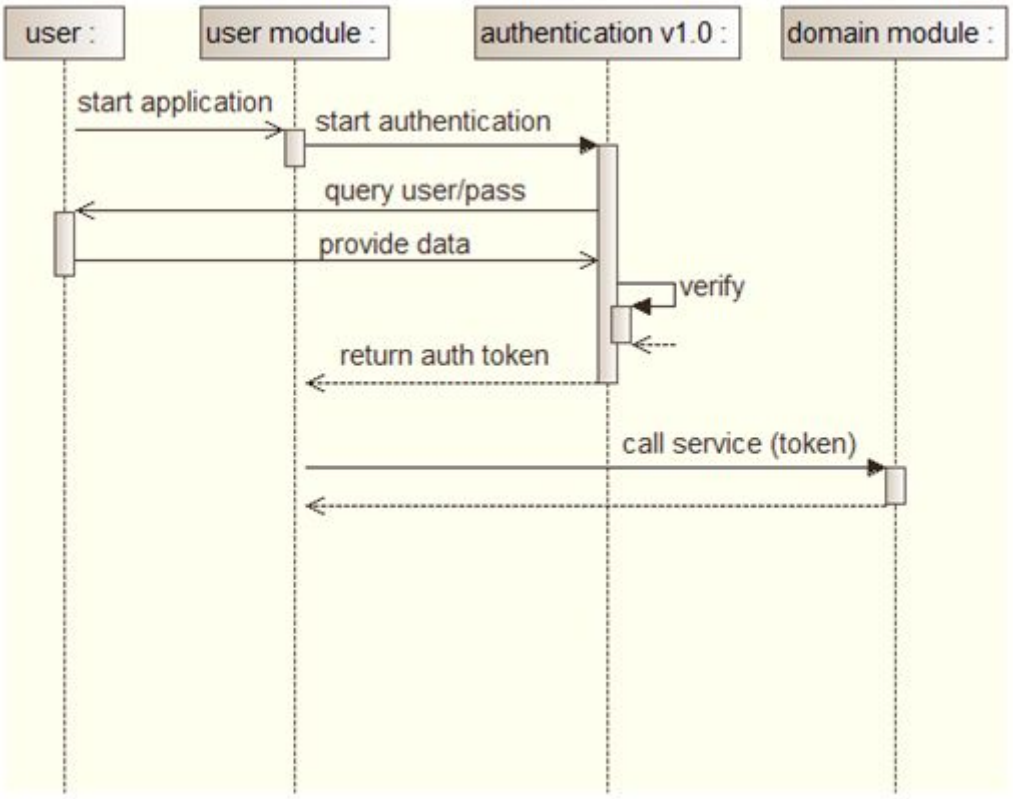
TODO for architecture documentation
by module developers using cross cutting API::

- Specify requirements for the service
- Document the decision between the three scenarios stated above
- Document the usage/change/implementation of the service according to the scenario list above.

If the service is changed by the project, the changes have to be documented in the services documentation.
If the service is newly defined, this has to be done in a separate project, and will be documented in that project.

The following three simplified examples show, which case falls in which scenario (here exemplarily for authentication (auth) if an ok authentication service exists):

Scenario 1:
REQ1: This project requires a user/password based authentication.
REQ2: Usernames shall be 6-10 characters
REQ3: Passwords shall be 8-20 characters
REQ4: If correct user/password is entered, an auth token shall be returned to the client.
Verdict from project and AC/QC: "Technical Service 'Authentication v1.0' according to its documentation v1.0 offers all required functionality."
Usage:



Scenario 2:

REQ1: This project requires a user/password based authentication.

REQ2: Usernames shall be 6-10 characters

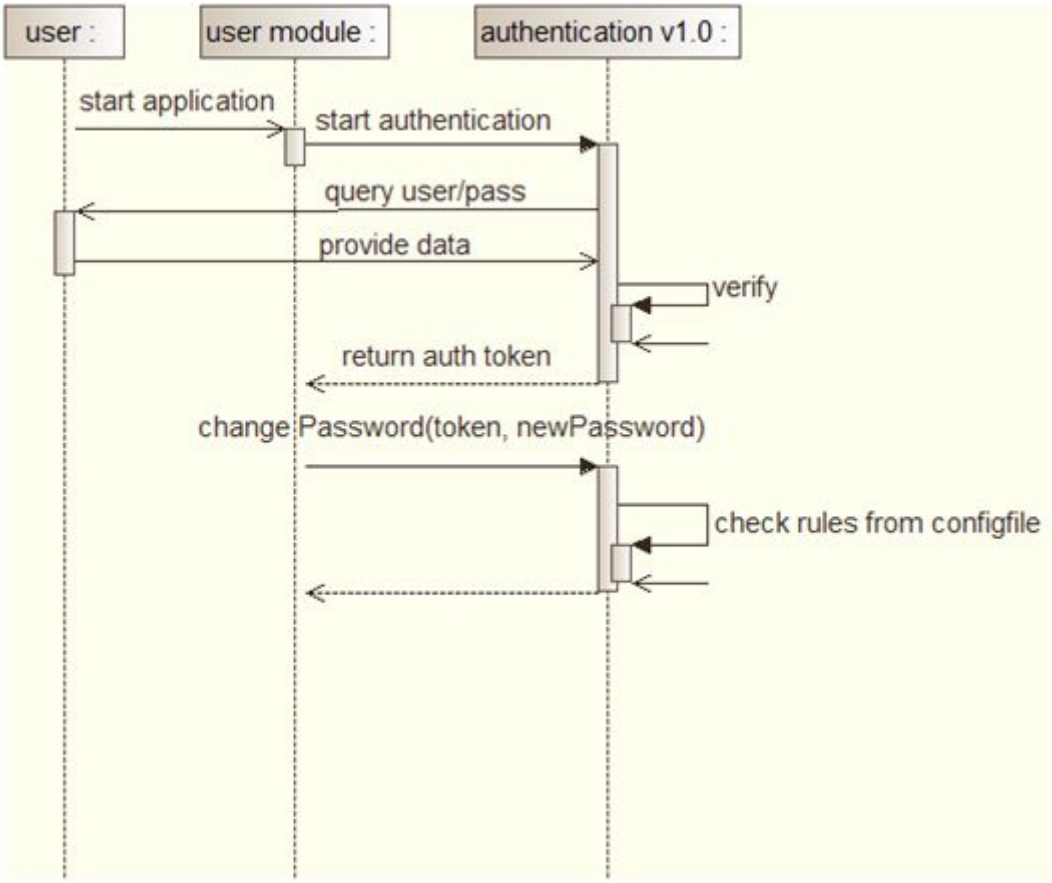
REQ3: Passwords shall be 8-20 characters

REQ4: If correct user/password is entered, an auth token shall be returned to the client.

REQ5: Password patterns (e.g. 'Password shall contain lowercase and uppercase characters and digits') shall be configurable.

Verdict from project and AC/QC: "The implementing open source component behind the 'Authentication 1.0' API is able to handle custom rules as required. The auth/auth core module has to be extended in order to read the rules from a configuration file, and to evaluate and enforce them during password modification. No new core module is required."

Usage:



Scenario 3:

REQ1: This project requires a user/password based authentication.

REQ2: Usernames shall be 6-10 characters

REQ3: Passwords shall be 8-20 characters

REQ4: If correct user/password is entered, an auth token shall be returned to the client.

REQ5: Password patterns (e.g. 'Password shall contain lowercase and uppercase characters and digits') shall be configurable.

REQ6: As an alternative to REQ1, usage of openID provider and hardware-based authentication (Smartcard or Bluetooth token) shall be possible.

Verdict from project: "The open source chosen to implement 'Authentication v1.0' is insufficient. A new core module has to be developed, which is able to collaborate with openID provider systems".

Verdict from AC/QC and PPC: For future re-use add the following requirement

REQ7: Biometric authentication shall be possible.

Then implement a prototype.

Table of core modules

Purpose	API doc link	Implementation	Comments
Authentication	Keycloak[1]	Keycloak	Implementation Module selected by SC & AC/QC
Authorization	Keycloak	Keycloak	Implementation Module selected by SC & AC/QC
Logging	Not yet defined	Not yet defined	

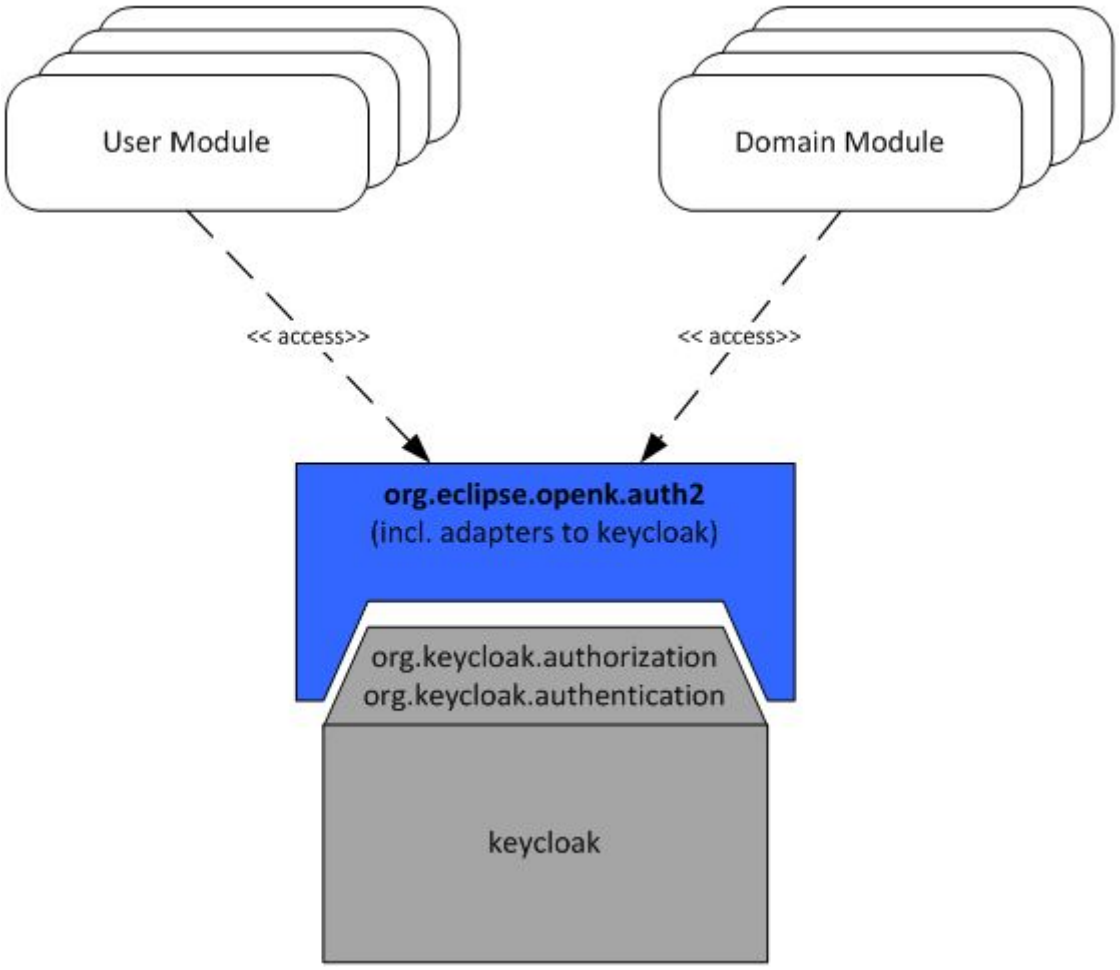
Please note that the first core module selected is keycloak for Authentication and Authorization. The openKONSEQUENZ comittees (SC, AC and QC) have selected keycloak as an implementation module based on an estimation of the requirements and past experience using keycloak. The API definition needs to be done in the first implementation project.

Auth²

Auth² is the core module implementing authentication and authorization protocols. *Keycloak* (http://www.keycloak.org/) is used as an implementation component.

As a shortcut, Auth² will currently be accessed directly via the Keycloak API. This will be replaced by **org.eclipse.openk.auth2**, an oK specific Auth² API shortly. The oK Auth² API will be an abstraction of the Keycloak API and is put into place due to possible migration and maintainability.

[1] Currently, keycloak is not integrated in any module. It shall be used indirectly in a core module. The first implementation project has to create an abstraction to thi Keycloak API in order to decouple keycloak and be able to switch the implementation in the future.



Logging

Next Core Module...

Domain API

Eisman

The Domain Modules known as “CIM Cache”

In the following, the three Domain Modules "TopologyDataManagement", "AssetDataManagement" and "MeasurementDataManagement" will be described. These domain modules are sometimes summarized under the term "CIM Cache" and have a central role in the openKONSEQUENZ platform, as these modules hold the integrated data model for higher level tasks for operating power grids. The CIM-standard-based interfaces are used to integrate and export data into and from the module. The major purpose of these CIM cache modules are to provide a shared data repository for faster, more directed and simplified access to data that is distributed and fragmented over several Source

Systems, that have in some cases low availability (e.g., a GIS might have an availability of 97%, which might be sufficient if it is used only as documentation system).

Especially these domain modules will be subject of regular future extensions, because future user modules might require data that is not yet in the data model. Such extensions may modify the interfaces (e.g., by adding attributes and entities) and possibly add new interfaces. Additionally, the current descriptions are prior to implementation and may change during implementation and further connectional architectural discussions.

In a concrete installation, only the interfaces need to be implemented that are required by the user modules that the DSO wants to use. Furthermore, there might be even more domain modules that are part of the CIM Cache module group, such as a module for caching events.

The development and APIs of the CIM Cache domain modules starts by focusing on the feed-in management use case, which is subject of the first user module and based on plans for a basic power grid state estimation module (which distributes eclectic power measurements to lower level elements based on profiles). The feed-in management use case is a good starting point, because it combines static topology, dynamic topology, asset management data and current power measurements. These use cases need a data model that can be used to answer the following (here simplified and generalized) questions:

- Which power generators (e.g., PVs or wind parks) are connected to which power transformer under the current switching state of the power grid?
- Which (HV/MV) power transformers are in the power grid?
- What is the current (i.e., last) active power measurement for a power generator (if this power generator provides measurements)?
- What are the measurements by sensors that are installed in the power grid (e.g., sensors located at power transformers, near to feeders or at secondary substations)?
- How are (primary and secondary) substations, power generators, feeders, consumers, generators topologically related (i.e., electrically connected)?

The three domain modules cut the CIM Cache into modular parts based on the decision that:

- different structural types of data (e.g., topology graph, lists of asset data, time series data) each have its own optimal data management strategy,
- to have independently deployable modules, since not every DSO will need all parts of the CIM Cache, and
- the CIM Cache is cut into parts depending on the semantic type of data (e.g., static topology that changes only by grid construction, dynamic topology data such as switch positions, asset management data of feed-in generators and grid infrastructure, and measurement data that changes frequently).

However the separation of the CIM Cache in multiple modules still means that the data is connected between the modules.

TopologyDataManagement (TDM)
This domain module manages the current topology of power grids. Later, it might also contain potential and planned (switching state) topologies, but not in the first implementation steps. This includes logic to build up topology data models from the source system APIs and logic to provide the data and to answer certain questions (e.g., is a grid element electrically connected to a power transformer).

The TDM includes:

- static topology information with grid infrastructure elements, such as wires, switches, transformers with some master data and their topological context
- dynamical topology information, such current switching and (later) tap-changer switching state
- current or recent measurement data of grid sensors (e.g., from sensors in substations)

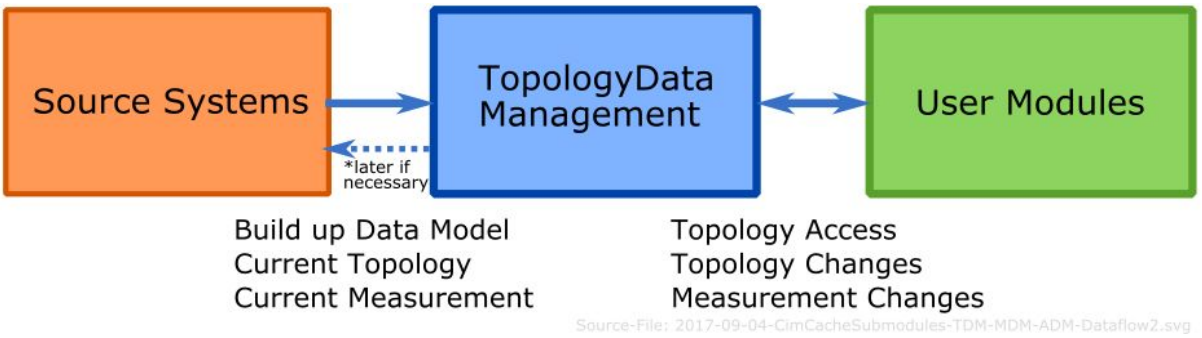
The TDM does not include:

- no complete master data for assets
- no measurement data history (i.e., it is not an archive / historian)
- no forecast data
- no real time measurements
- no planned switching

The TDM provides services for:

- requesting the complete topology and for parts of the topology
- starting a complete rebuild of the topology data model
- refreshing static topology, dynamic topology or measurements
- subscribing to changes to static and dynamic topology
- subscribing to changes of measurement values

The interfaces should as far as possible be implemented as standard CIM profile. For certain tasks, industry standards such as OPC UA/CIM could be more suitable. The data flow for TopologyDataManagement is shown in the following figure. Currently, data only flows from source systems to TDM. Later, a data flow from TDM to source systems could be established after careful consideration of architectural consequences and additional approval of the AC.



The TopologyDataManagement might use Core Modules in the future for tasks such as authentication/authorization, logging, and technical monitoring.

AssetDataManagement (ADM)

The AssetDataManagement temporally manages a subset of the master data of grid infrastructure (e.g., transformers) and grid connected devices such as decentralized energy resources. This master data focuses on major electrical properties and on some non-electrical properties that are relevant for operational tasks (e.g., contact information for a decentralized energy resource). This module should not be confused with a general Asset Management System - this module holds a copy of some of the data in the Asset Management System.

The ADM has interfaces for:

- storing master data of grid infrastructure elements and grid connected devices
- refreshing of the master data
- requesting (total or parts) master data of (all or some) of the grid elements

Any entity (e.g., a transformer or decentralized energy resource) can have a reference to the topology (via its unique identifier). The data flow for AssetDataManagement is shown in the following figure. Currently, data only flows from source systems to ADM. Later, a data flow from ADM to source systems could be established after careful consideration of architectural consequences and additional approval of the AC.

The AssetDataManagment might use Core Modules in the future for tasks such as authentication/authorization, logging, and technical monitoring.

MeasurementDataManagement

The MeasurementDataManagement stores selected time series data from the power grid and from consumers and producers that are connected to the grid. It stores measurement data together with their location in the topology and provides access to this data. It is not an archive of the complete available measurement data from the OT-Layer (operational technology layer, e.g., SCADA & DMS).

This Domain Module is not required for the current feed-in management use cases and a simplified state estimation use case. Therefore, the APIs will be specified at some time later.

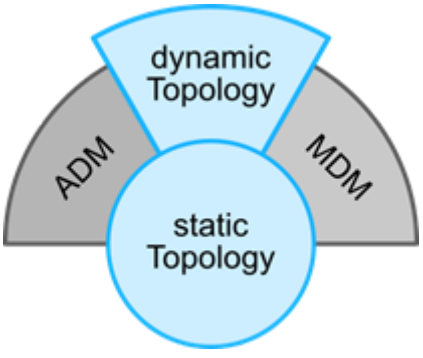
The MeasurementDataManagement has interfaces for:

- Creating and initial storing of relevant power system measurement data with its location in the topology.
- adding new time-value-pairs to existing time series
- accessing time series in a grid area or for a certain grid element

The data flow for MeasurementDataManagement is shown in the following figure. Currently, data only flows from source systems to MDM. Later, a data flow from MDM to source systems could be established after careful consideration of architectural consequences and additional approval of the AC.

The MeasurementDataManagement might use Core Modules in the future for tasks such as authentication/authorization, logging, and technical monitoring.

Domain-Module Dependencies



The figure above shows the dependencies between the described domain-modules ADM, MDM and TDM. The TDM (blue) consists of two services. The core-service provides static topological-information about the installed conducting equipment. This information can be used by the second TDM-service (dynamic Topology), that uses the current switch- and tap-changer-states, to enable the view on the current electrically connected power grid. The

ADM, MDM and TDM-dynamic Topology can reference equipment, using the unique identifier that is provided by the TDM-static Topology-service.

Source-System-APIs

To get data from source systems, such as DMS, GIS, ERP, oK defines source system APIs. These APIs are developed under the principle of separation of concerns and described using Swagger. The current status can be found in the ok-API Documents.

5.2 Level 2

TODO for architecture documentation
by module developers according to Building Block View Level 2:

Document Level 2: Internal components of a module and a logical view have to be documented as component diagram and class diagram with UML:

- Overview of components and internal interfaces
- Detail specification of “complex’” (according to QC-Handbooks top 2 categories) modules as UML component and class diagram
- Detailed documentation as JavaDoc
    - Separation between architecture and quality is needed in order of the use and especially the maintenance of modules
- Database model as entity relationship or UML class diagram
- List existing and new Access rights, that are required by module (and how to create them).
- Document security relevant system components and implementation specification (according to BDEW Whitepaper chapter 2.1.2.1, if it has to be applied).

6. Runtime View

Todo for architecture documentation
by module developer according to chapter 6:

- Show Interaction of building blocks (components) at runtime for exemplary processes (use cases / scenarios) with bpmn diagrams (for functional processes) and sequence/collaboration UML diagrams for technical processes.
- Non-trivial processes in the business logic and processes, in which several modules are involved, have to be modeled as UML sequence or collaboration diagram.
    - Several modules scenarios:
        - Technical documentation of the processes as sequence diagram, which shows calls between user, modules and external systems.
        - Matching of interfaces of the modules according the functions- or service calls for quality assurance
    - Module intern scenarios:
        - UML sequence diagram (or collaboration diagram) define functions-, communications- and data-flows for a scenario

7. Deployment View

The module developer must develop system interface mockups (https://en.wikipedia.org/wiki/Mock_object) for the external systems or legacy systems that the developed module needs data from. These mockups shall be integrated in deployment stages as early as possible, but must be available and integrated in Quality Assurance Environment for Sprint Reviews. If there is a demand in the tender also for integration of real systems, one of the DSOs (specified in tender) has to provide appropriate test systems for integration.

There are several deployment environments required for the different participants and roles. These deployment environments are based on a reference environment where the reference platform software is preinstalled. Code check, testing and build/integration/deployment tools (etc.) according to QC Handbook will be made available on the reference system and therefore be part of the deployment template accessible for the developers.

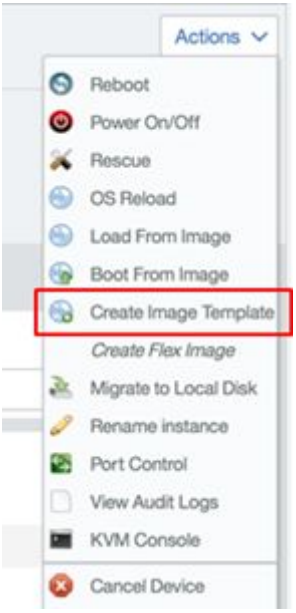
The different deployment environments are listed in following table and then detailed:

Environment Name	Short description	Logical order of	Reference Environment	Responsibility	Module developer's

		deployment	usage (Y = yes - mandatory, N = no, P = possible)		task
Reference Environment	The oK standard environment used as reference and the oK overall test environment	4	Y	oK	Support deploying their modules
Develop Environment	Environment a developer is using for debug & test purposes	1	P	Developer	Up to the developer
Integration Environment	Environment for checks of integration of branches	2	P (recommended)	Developer	Up to the developer
QA Environment	Environment for Sprint Reviews	3	Y	oK	Deploy module & legacy system mocks; Product Owner acceptance
Demo Environment	Environment for demonstration of oK to “the world”	5	Y	oK	Integrator or Developer (decision of SC/PPC)
Customer Environment	DSO Environments for integration in DSO system landscape, test, and	6	P (recommended)	DSO	-

	production				

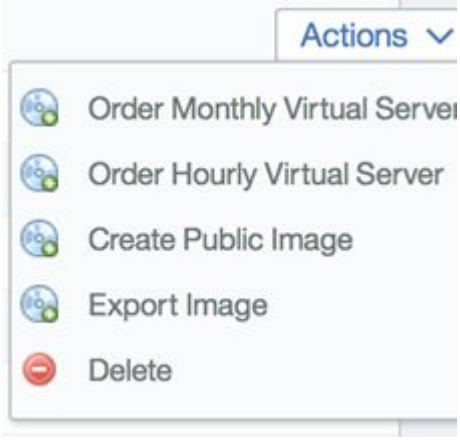
- Reference Environment - The reference environment is the reference instance for all other deployment environments and is operated out of an IBM Softlayer Cloud in Frankfurt am Main.
    a. The reference system has the latest accepted version of the oK reference platform and the modules.
    b. Updates to the reference system can be made after successful test and acceptance in a staging environment that is different from the reference environment.
    c. The reference system is an 'on demand' system.
    d. The creation of a template from the reference system and storage of the template in the library allows for easy instantiation for the various other deployment environments -- test, dev, acceptance, ... The creation of a template can be done easily through the cloud management portal environment:



    e. The current configuration of the reference system is based on the experience from the pilot and will be changed according to the progressing needs.
    f. This system is owned by openKONSEQUENZ
- Quality Assurance Environment - is the acceptance environment for sprint and module acceptance through the SCRUM product owner and the Lead Buyer. The module needed data of legacy systems must be available as a mockup. Once accepted, changes can be deployed into the reference system. The quality assurance environment can either be 'permanent' or 'on demand' and is owned by openKONSEQUENZ. A quality assurance environment can easily be deployed using the reference system template within the cloud environment. Using the cloud

environment specific adaptations to the hardware requirements (like cores, memory or disk) can also be validated within this environment.

- Integration Environment - is up to the developer but should be near to the quality assurance environment to ensure the same characteristics. The developer can also make use of an IaaS platform and template to deploy a template of the openK reference system. The developer is asked to have his own account to deploy the reference template within the IaaS platform used by openKONSEQUENZ. To make it very easy for every application developer and contributor to access a copy of the reference system for integration test purpose the IBM Softlayer portal allows for easy use of the defined reference system templates.

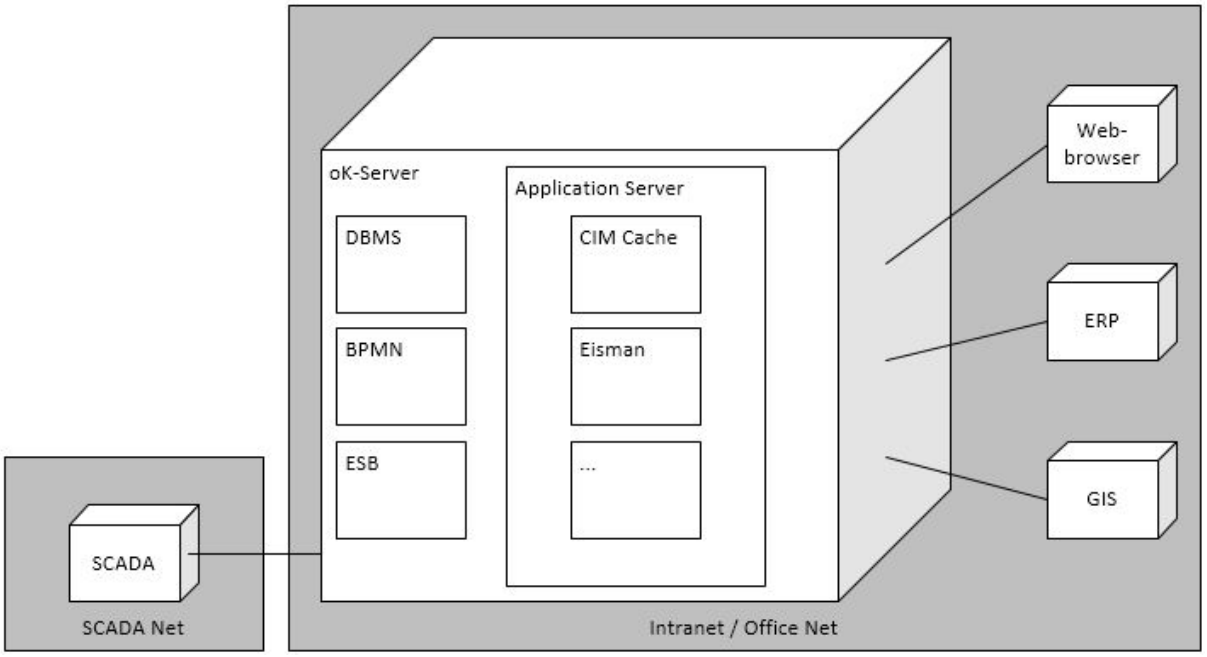


  This integration system possibility from the cloud allows the application developer to run performance tests and adapt the system environment to meet the requirements and provide guidance to openKONSEQUENZ for system sizing related to the newly designed and developed application.

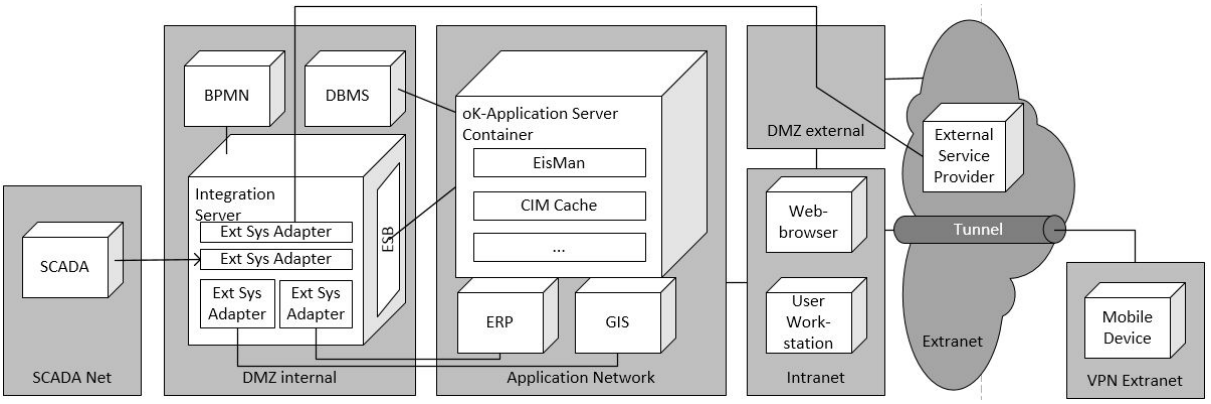
- Demo Environment - it is planned to set up a demo environment including existing modules. This demo environment will be accessible from external (public IP-address) and is equivalent to the reference system incl. interaction between the modules and legacy system mocks. The demo system should be a permanent system and is owned by openKONSEQUENZ.

The reference environments associations to legacy systems are shown in the following figure. For quality assurance environment and demo environment, the external data sources

(examples the figure are SCADA, ERP, GIS) are integrated in the oK-Server as system mocks.



- Standard Customer Environment - there is not standardized way, how DSO are splitting up resources on multiple hardware components (see following image for a possible scenario)



It is possible, that the SCADA System has access to the integration server, but not the other way around.

The architecture principle is based on a service oriented architecture. The decision was taken not to adopt a micro services architecture at this point in time. This may become more relevant in the future when more and more Platform Modules and / or applications provide reusable components.

Recommended information for developers according building up an own infrastructure or using the reference environment in an own hosted VM or on the IaaS

The openKONSEQUENZ reference system is hosted in an IaaS (Infrastructure as a Service) environment with IBM Softlayer Cloud in Frankfurt/Main.

Image templates of the reference system will be created and made accessible.

Application developers will get access to the reference system templates. A developer can decide to use an own infrastructure, the .vhd format-templates in a virtual machine or the IaaS environment. In the latter he can easily deploy these templates for development and or testing purpose - in the case of using the IaaS environment it is typically in the responsibility of the application developer to have his own IaaS account and access. It is recommended to use at least the template in the integration environment.

TODO for architecture documentation
By module developer according to chapter 7:

- Document how the modules builded source is deployed in the reference environment to get the module started
- Document the external or legacy system mockups (name, which data, which system(s) the data usually will come from, which external interface is used (name, version), mockups git-repository position).
- Document requirements and assumptions needed for secure system operation according to BDEW Whitepaper (if it has to be applied) chapter 2.1.2.4

8. Concepts and Non-Functional Requirements

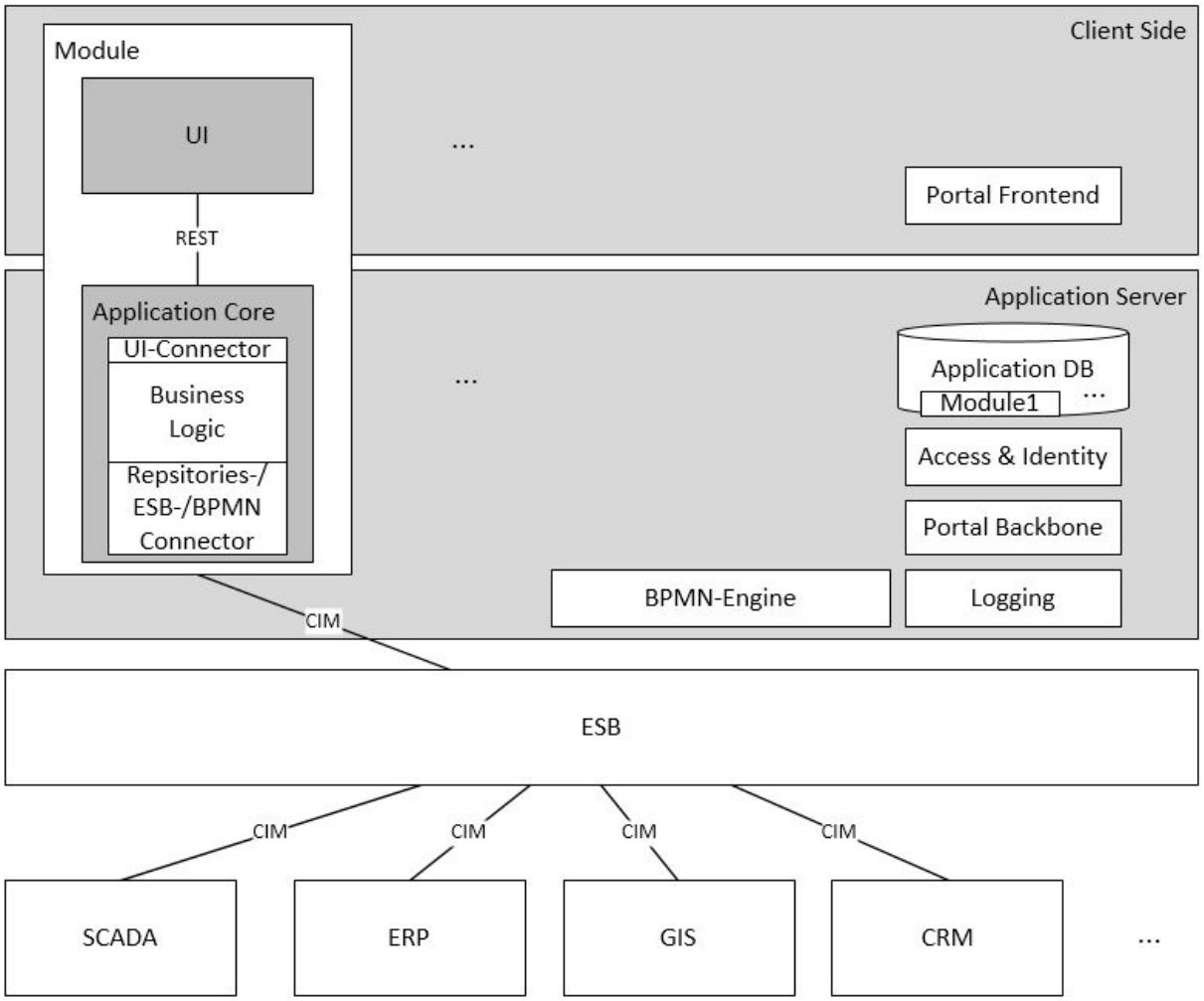
This chapter describes the general concepts and for this architecture handbook relevant detailed information - irrelevant subchapters remain to keep consistency according to the Arc42.

General concepts are:

Tiers (see also following image):

- Client-Side GUI
- Business logic on application server with interface sub tiers to client, ESB/Business process engine, and module specific persistency (The business layer (if no BPMN-process) of an application must be placed in the Java application. This layer shall explicitly not be only a pass-through layer. It needs to hold its own business objects to get separation between ESB and GUI.)
- Business process engine as possible tier between ESB and application server (Camunda is suggested by openKonsequenz AC for BPM based applications. Current applications do not yet require a BPM engine. This may change in the future. The AC is open to discuss alternative suggestions form application developers.)
- ESB
- Legacy systems adapter

- Legacy systems



Data-Exchange:

- Modules interact only via ESB and CIM interfaces (web services, REST, XML, CIM/XML (RDF)) with other oK modules or external systems.
- Each module must store its own entities in private module-DB(-scheme) (there is no data exchange between modules using the same DB scheme allowed)
- Communication between different modules or legacy systems only via ESB
- CIM-based ESB communication (IEC 61970 / IEC 61968 / IEC 619325)

Vendor-neutrality:

- Use of Open Source libraries
- Special features of ESB (and other middleware) are not used to prevent vendor-lock-in

User Interface:

- ok-GUI-Styleguide based on previous work of Minnemedia and openK pilot
    - oK-Design of minnemedia contains
        - Styleguide,
        - Html-pages
        - Bootstrap artefacts

- JS artefacts

Development/Quality/Test:

- Development chain according to Quality Committee Handbook

In Future: Identity and Access Management (in development)

In further subchapters, details of the architecture concept are listed or have to be listed and documented for each individual module. Empty 8.x subtitles are not of importance for AC handbook but may be of importance for individual module documentation.

8.1 Domain Models

The interface models of the overall openKONSEQUENZ platform interface and of module interfaces shall be shown under 3.3. External Interfaces.

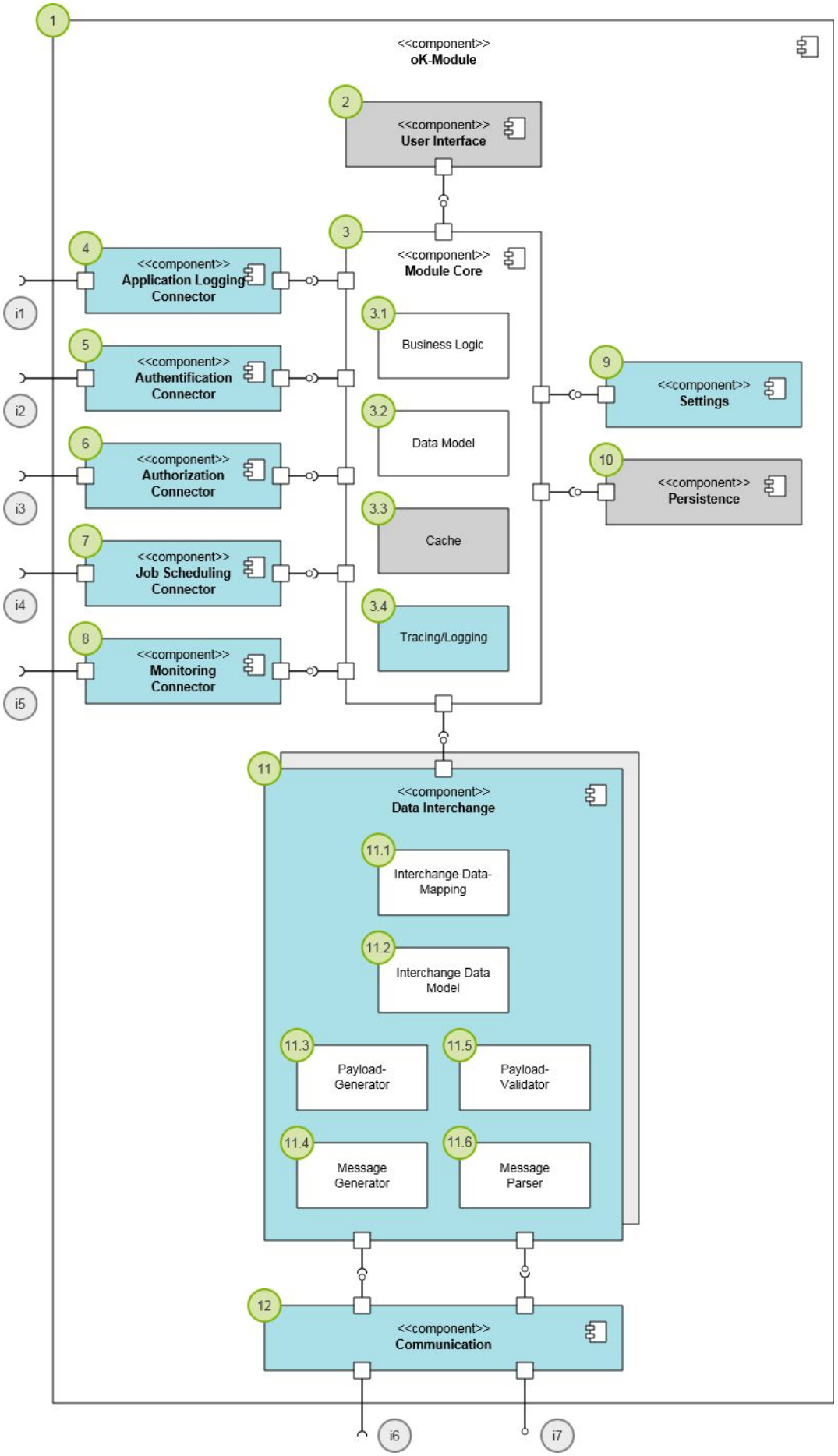
TODO for architecture documentation
by module developer according to Chap. 8.1:

- Domain models for the business logic of modules without relation to technology have to be documented at this point by each module developer.

8.2 Recurring or Generic Structures and Patterns

Internal module architecture

In the following, an example architecture for the internal architecture of a single oK Modules is described. This description can be used as template and it is currently not mandatory (ACQC Meeting 2016-08-29). The following UML component diagram (https://en.wikipedia.org/wiki/Component_diagram) shows the proposal for an oK-Module:


1
<<component>>
oK-Module
2
<<component>>
User Interface
3
<<component>>
Module Core
3.1
Business Logic
3.2
Data Model
3.3
Cache
3.4
Tracing/Logging
4
<<component>>
Application Logging Connector
5
<<component>>
Authentification Connector
6
<<component>>
Authorization Connector
7
<<component>>
Job Scheduling Connector
8
<<component>>
Monitoring Connector
i1
i2
i3
i4
i5
9
<<component>>
Settings
10
<<component>>
Persistence
11
<<component>>
Data Interchange
11.1
Interchange Data-Mapping
11.2
Interchange Data Model
11.3
Payload-Generator
11.5
Payload-Validator
11.4
Message Generator
11.6
Message Parser
12
<<component>>
Communication
i6
i7

1 oK-Module

The oK-Module component consists of different components and parts. Some of them have specific functional requirements and must be implemented especially for the application purpose. But there are also components and parts, which have common functionality and features in the oK-context. Some components and parts are not necessary and can be omitted . They are optional.

- gray components and parts are optional
- blue components and parts are candidates for a common oK-framework library

2 User Interface (optional)

The User Interface is an optional component for modules that interact with users. Usually the Domain Modules have no user interface. The User Interfaces are web-applications, that are connected via a REST/JSON communication with the backend.

3 Module Core

The entire application logic is provided by the Module Core. It defines the flow control and manages the interaction of its components and parts to fulfill the functional and technical requirements.

Business Logic:
This part is the heart of the Module Code. It defines the functional requirements.

Data Model:
This is the application specific data model.
It can be inspired by the CIM standard. But be careful. Usually it is not recommended to use the CIM-Model as application own data model. It has to be attentively evaluated, if your performance and software design requirements can be fulfilled. Please also keep in mind that you have a potentially unwanted dependency to CIM.

Cache (optional):
For performance reasons it is sometimes a good idea to hold often used application data in the memory. The application can use the optional Cache component for this purpose.

Tracing/Logging:
The Tracing/Logging component records technical and functional events, messages and errors that occur while the application is running. It can be used to understand or reproduce malfunctions. It also can be used for archiving purposes and audits. Usually a local-file and/or application console keeps the trace and log records.
To create homogeneous formed log entries and use comparable log levels, it is recommend to implement and use a common oK logging component.

4 Application Logging Connector

The Application Logging Connector provides access to the Application Logging Module of the oK-platform.

5 Authentication Connector

The Authentication Module can be used for authentication purposes. This component provides access to this module. It should be implemented module independent, so it can be reused by other oK-Modules.

6 Authorization Connector

The Authorization Connector is similar to the Authentication Connector and provides access to the common Authorization Module.

7 Job-Scheduling Connector (optional)

Some oK-Modules provides time-dependent processing. With the Job-Scheduling Module it is possible to control the time-dependant processing of different oK-Modules in the right chronological sequence. If the current oK-Modules has any time-dependant processing, the Job-Scheduling Connector enables the access to the common Job-Scheduling Module.

8 Monitoring Connector

Using the Monitoring Module the system operator gets the accurate status information of every distributed oK-Module at one glance. To enable this functionality, it is necessary that every oK-Module provides information about its current state. The Monitoring Connector provides the access to the Monitoring Module.

9 Settings

Normally oK-Modules have a set of configurable values, that are not changed throughout the entire module runtime. For example, they enable the integration of a module in different environments or enable/disable optional module features. The component Settings provides the current configuration. Since the way to access and store settings can be reused, it is recommend to implement or use a common component of the oK-framework library.

10 Persistence (optional)

With the optional Persistence component, the oK-Module can persist its data. Usually a database management system is used for this purpose. But the most suitable storing method depends on the specific needs and requirements of the actual oK-Module.

11 Data Interchange

The Data Interchange component is used to exchange data and information with other oK-Modules or the source systems.

Since the communication with the Module Core is encapsulated in an interface, the Data Interchange component can be replaced easily. So it is possible to minimize the costs, when

the communication standard or version will be changed. It is also possible to operate different communication standards or versions concurrently.

Interchange Data Mapping:
The function of this part is to map the Data Model of the Module Core to the Interchange Data Model and vice versa.

Interchange Data Model:
The Interchange Data Model is the representation of the data that should be exchanged with other oK-Modules or the Source Systems. Usually, but not necessarily, this is a CIM-Data Model.

It is an indicator for bad software design, if you use the Data Model as Interchange Data Model. The two models are usually different and have to be maintained independently.

Payload Generator:
This part creates the functional information, that should be transferred to the receiving oK-Module or Source System

Message Generator:
The Message Generator prepares the message for transferring. It enriches the payload with meta and routing information that enable the messaging middleware to forward the message and help the receiver to parse the payload.

Payload Validator:
The Payload Validator checks the functional payloads of the incoming messages, if they meet the minimum defined requirements.

Message Parser:
Incoming messages will be analyzed and parsed by the Message Parser. The Message Parser unpacks the payload of the message and transfers it to the Payload Validator.

12 Communication
The reusable Communication component receives or sends the data, that should be exchanged with other oK-Modules. It complies and implements the guidelines that are defined in the Communication Guideline.

8.3 Persistency

Java Persistence API (JPA) reference implementation EclipseLink with PostgreSQL (without using special features for each of them to make a potential exchange of database and JPA easy). In the sharing PostgreSQL each module shall have its own database schema or more

than one. Different modules are not allowed to access the same database schema. The module name must be used as prefix for the database schema names and for database users.

8.4 User Interface

The ok-GUI-styleguide document defines rules and regulations that each module has to follow, when it provides user interfaces. The GUI-styleguide document is evolving, and will be extended as required. If the GUI-styleguide does not cover UI concepts needed by a module, it shall be extended during the module development. The project may suggest an extension to the GUI-styleguide document, but additions to and changes of the GUI-styleguide document have to be reviewed, accepted and released by the architecture committee.

8.5 Ergonomics

See 8.4

8.6 Flow of Control

Intentionally left blank - not relevant at this time.

8.7 Transaction Procession

Intentionally left blank - not relevant at this time.

8.8 Session Handling

User Session Handling: User login and session management is provided by the portal (Liferay in case of oK). All modules must use the user session management of the portal to achieve an uniform experience for the user and to ease integration into DSOs environment. User session management is standardized in JSR 286 portlet specification.
If a module needs management of technical sessions (e.g. sessions with external systems, or involving multiple services), these aspects have to be described here.

TODO for architecture documentation
by module developer according to Chap. 8.8:

- Document the technical sessions management (if needed)

8.9 Security

According to the vision of openKONSEQUENZ (see document oK-vision) the oK-software must be secure. This chapter will contain the security concept. The concept will define the implementation of the requirements of the BDEW Whitepaper and will be aligned with its

structure. For the security relevant aspects which are laid down at other locations it will contain references. With increasing amount and functionality of user modules and Domain Modules the security concept will be extended.

General Requirements and Housekeeping

Cryptographic standards

When cryptographic standards are selected, only state-of-the-art cryptographic algorithms and key lengths according to BSI TR-02102-1 shall be used. **This is particularly true for the usage of** Transport Layer Security (SSL/TLS) in connections with HyperText Transfer Protocol Secure (HTTPS) - see BSI TR-02102-2.

Documentation

A complete security architecture does not only comprise technical means. It also describes operational guidelines considering the available technical base as well as the personnel controlling the systems. The Documentation of Security Parameters and Security Log Events or Warnings has to be described according to chapter 5.2, 8.16 and 8.18.

The end user documentation:

- User documentation
- Administration documentation

is demanded by the QC handbook and shall include the Requirements and Assumptions needed for Secure System Operation.

The security concept is not yet fully specified. A task of the AC is to extend the security concept according to progress of development. Further subchapters according to security shall underlie the following structure:

- Base System
- Networks / Communication
- Application
- Development, Test and Rollout
- Backup, Recovery and Disaster Recovery

8.10 Safety

According to the mission statement of openKONSEQUENZ the software is located in a safety-critical environment. Until further notice, the software will not be directly coupled with or responsible for functions that might endanger human life or equipment. Safeguarding life and environment is not in focus for the software.

8.11 Communications and Integration

Intentionally left blank - not relevant at this time.

The integration concept is yet not considered to a sufficient extent. A task of the AC is to define a concept for replacement / integration of modules.

8.12 Distribution

Intentionally left blank - not relevant at this time.

8.13 Plausibility and Validity Checks

Intentionally left blank - not relevant at this time.

8.14 Exception/Error Handling

Exceptions that are part of a modules external interface need to be discussed with the AC and cleared by it. Each module is responsible for its own error/exception handling as specified in the list of technologies below. All errors and exceptions shall be logged according to the logging requirements as specified in the QC Handbook.

TODO for architecture documentation
by module developer according to Chap. 8.14:

- Document which kind of exceptions and errors are handled by the system
- Document **Which kinds of errors are forwarded to which external interface** and which are handled fully internally.

8.15 System Management & Administration

Larger software systems are often executed in controlled environments (data centers) under oversight of operators or administrators. These stakeholders require specific information on the applications' states during runtime as well as special means of control and configuration. In oK, the development processes are separate Processes. On the one hand, oK development of prototypes and on the other hand the bilateral system integration at the DSO. Because of this, the need for a proper implementation at this point is even stronger. The system administrators need information, where and how ticket management systems can get fault information.

The system management & administration concept is yet not considered to a sufficient extent. A task of the AC is to specify a central instance for getting tickets for administration of oK-Platform.

TODO for architecture documentation
by module developer according to Chap. 8.15:

- Document where and how a ticket management system can get fault information.

8.16 Logging, Tracing

There are different kinds of logging. Logging for business aspects, logging for administrators, logging for developers. It is specified in the tender / by Product Owner, which events must be logged for business aspects.
The Logging shall be implemented using the Simple Logging Facade for Java (SLF4J). As Implementation for the Logging, for example Log4J, Log4J 2 or logback can be used.

TODO for architecture documentation
by module developer according to Chap. 8.16:

- What are expected messages, which meaning does they have and (if necessary) how a module differs from QC-requirements.
- Document security relevant system messages (according to BDEW Whitepaper chapter 2.1.2.3, if it has to be applied).

8.17 Business Rules

Intentionally left blank - not relevant at this time.

8.18 Configurability

- Modules have to be configurable in that way, that no rebuild is required to run code in different environments (see chapter 7. Deployment View for different environments - in especially different operating systems and distributed servers with different access rights).
- Module specific configuration has to be done in one module-central file. All configuration parameters shall have meaningful default values. The semantics, value ranges, and interdependencies of all configuration parameters shall be documented as a part of the modules architecture description.

TODO for architecture documentation
by module developer according to Chap. 8.18:

- Check/develop module dependencies according to different environments
- Document configuration files/properties
- Provide configuration file for quality assurance environment
- Document security relevant configuration (according to BDEW Whitepaper chapter 2.1.2.3, if it has to be applied)

8.19 Parallelization and Threading

Intentionally left blank - not relevant at this time.

8.20 Internationalization

By default, all user interface elements shall be internationalized. I.e. all strings, colors, number/date formats, fonts, ... need to be configurable. Each module has to provide a german nationalization.

8.21 Migration

Intentionally left blank - not relevant at this time.

8.22 Testability

See QC handbook. Projects should support at least daily build-and-test cycles. Important keywords for this aspect are unit tests and mock objects.

8.23 Scaling, Clustering

Intentionally left blank - not relevant at this time.

8.24 High Availability

Intentionally left blank - not relevant at this time.

8.25 Code Generation

Intentionally left blank - not relevant at this time.

8.26 Build-Management

See QC handbook.

TODO for architecture documentation
by module developer according to Chap. 8.26:

- Document, how the overall system is created from its (source code) building blocks. Document, if directories are in line with the given structure in QC handbook (Which repositories contain source code, where are configuration files, test cases, test data and build scripts maven stored).

8.27 Offline-Module

Some future modules may have a non-functional requirements to work also offline, that have to be kept in mind in earlier design decisions for the whole platform. In some cases, workers in the field may need modules functionality even if their device is offline (from IT-Network (e.g. there is no active connection to the internet)). Module developers only

need to implement these requirements, if it is explicitly requested in the PPC module call for tender / definition.

9. Design Decisions

Design and technical decisions are listed in the following chapter. Documentation of design decisions is useful for better understanding of existing architecture and shall avoid non-observance of existing knowledge.
In case of deviation from the openKONSEQUENZ technical decisions by a module, the module developer has to ask the ACQC committee to come to an agreement.

1. Platform Components for more than one Module
    a. Problem: Several Modules need the same data/information.
    b. Constraints: Data/information shall not be saved redundant in every piece of software.
    c. Assumptions: Interfaces can be developed for exchange of such information
    d. Alternatives:
        i. Central platform modules
        ii. Distributed information in modules
    e. Decision: Central platform modules shall be developed for topology management. Also for Identity & Accessmanagement.
2. Module communication: Communication between modules only via ESB
    a. Problem: Modules need information from other modules
    b. Constraints: If one module changes, other modules shall not need a change.
    c. Assumptions: if modules interact directly via Databases, no module is directly responsible for the database scheme, schemes can be misunderstood or misinterpreted so inconsistency in the underlying data is supported.
    d. Alternatives: direct access to database, interface communication
    e. Decision: interface communication via ESB, because of better long term maintainability (independent Maintenance cycles between modules and separated interchangeability during operation)
3. No multitenancy
    a. Problem: Multiple DSOs could use one module
    b. Constraints: concurrent use of modules by different DSOs
    c. Assumptions: Every DSO has its own application server for hosting modules.
    d. Alternatives: multitenancy, no multitenancy
    e. Decision: No Multitenancy
4. Portal
    a. Problem: Users shall not need to register/login for each oK-module
    b. Constraints:
        i. Single-Sign-On or Workstation Login
        ii.

c. Assumptions: Liferay handles User Sessions, Konfiguration & Access Rights for Portal-pages.
d. Alternatives:
    i. Liferay
    ii. oK-self implementation
e. Decision:
    i. Liferay

5. Identity & Access Management
    a. Problem: Users shall be differentiated and have different access rights to modules.
    b. Constraints:
        i. An oK-self implementation is very cost intensive
        ii. Open source tools shall be used and abstracted via an API
    c. Assumptions:
        i. Liferay can be used to manage Rights & Roles not only for GUI, but also for backend.
        ii. Liferay can incorporate existing LDAP or AD user token
        iii. Keycloak can be used to manage Rights & Roles in the backend
        iv. Keycloak can incorporate existing LDAP or AD user token.
        v. The use of Liferay may hinder replacement in later point of time
    d. Alternatives:
        i. Liferay
        ii. Keycloak
        iii. Ok-self implementation
    e. Decision:
        i. Keycloak must be used for Rights&Role Management to enable Access & Identity Management with a facade (that needs to be developed to make Keycloak replaceable in oK context)
6. ESB Talend vs. Mule
    a. Problem: Which ESB to choose
    b. Constraints:
        i. No vendor lock in, open source
        ii. ESB only transport medium, no higher functions shall be used to prevent lock-in.
        iii. Choice of ESB applies only for the reference platform - the DSO may use others in their production environment.
        iv. Exchangeability must be ensured
    c. Assumptions:
        i. Open source ESB prevents vendor-lock-in and can be developed further, Talend can be used in long term.
        ii.
    d. Alternatives: (By previously study) Talend, Mule

    e. Decision: Talend (got feedback on request)
7. ESB CIM/REST Interfaces
    a. Problem: A founded way to describe DSOs data is needed
    b. Constraints: Standardized Base,
    c. Assumptions: RESTful Webservices can be used according to CIM Standards, REST is easier to integrate than Soap
    d. Alternatives: CIM in different Versions, RESTful, Soap
    e. Decision: CIM Version 17 or newer, REST
8. BPMN Camunda (not explicitely specified)
    a. Problem: Process Engine
    b. Constraints:
    c. Assumptions:
    d. Alternatives:
    e. Decision: Camunda (It is suggested to use Camunda for business processes. If a software developer names an alternative, the AC will discuss this.)
9. Business reporting
    a. Problem: some Modules need to create human readable reports of business purposes
    b. Constraints:
    c. Assumptions:
    d. Alternatives: JasperReports Library, BIRT, Crystal Reports
    e. Decision: As Crystal Reports is closed source software, JasperReports Library or BIRT shall be used for generation of business reports.

    An inspection of licence compatibility to EPL of the listed report tools through eclipse is not done yet and needs to be triggered (possibly, by the first module utilizing reports).
10. Code-/Design-/Quality reports as .ad
    a. Problem: In which format shall reports for Code, design and quality be stored (and where)
    b. Constraints: Easy access (open community) - technologically and according storage
    c. Assumptions: git is used as repository for code. Ascii-Documents can be opened by everyone and easily be managed in git (version history, merge)
    d. Alternatives: Wiki, git, word, tex, pdf, .ad
    e. Decision: Ascii-Documents in git.
11. Name - Spaceholder
    a. Problem:
    b. Constraints:
    c. Assumptions:
    d. Alternatives:
    e. Decision

TODO for architecture documentation
by module developer according to Chap. 9:

- Each module has to describe its specific design decisions according to the following structure in its own architecture document:
    1. Name of decision 1
        a. Problem:
        b. Constraints:
        c. Reason for module specific and not global decision
        d. Assumptions:
        e. Alternatives:
        f. Decision:
    2. Name of decision 2
        a. Problem:
        b. Constraints:
        c. Reason for module specific and not global decision
        d. Assumptions:
        e. Alternatives:
        f. Decision:
    3. ...

10. Quality Scenarios

For quality KPIs and quality assurance we refer to the quality committee handbook, in which (in parts) formalized QA for continued tests is stated necessary. It gives a minimal set of test and acceptance rules and KPIs for unit tests.

The quality scenarios for the oK software are not yet considered to a sufficient extent. A task of the AC is to define hard specific requirements for the oK software, for the reference platform and quality scenarios for testing these requirements.

TODO for architecture documentation
by module developer according to Chap. 10:

- Each module has to describe its own quality requirements in quality (evaluation) scenarios and quality tree for sprint and final acceptance
- Test scripts and test handbooks for acceptance of sprints/releases have to be documented.
- Test datasets (on base of oK common test data - if it exists); have to be generated, published, coordinated with ACQC and used.
- According to the SCRUM Development process, when defining quality requirements it is also necessary to define acceptance criteria (the developer shall remind the product owner to discuss it in sprint planning).
- Commisioning Tests

11. Technical Risks

The technical risks for the platform are not yet considered to a sufficient extent. A task of the AC is to collect and assess risks for the reference platforms architecture.

TODO for architecture documentation
by module developer according to Chap. 11:

- Document the list of identified technical risks (with probability of occurrence, amount of damage, options for risk avoidance or risk mitigation), ordered by priority

12. Glossary

Short	Long (engl)	German	Description
AC	Architecture Committee	Architekturkomitee	Gives Framework and Constraints according architecture for oK projects
AC handbook	Architecture Committee handbook	Handbuch des Architekturkommitees	Textural guideline for module developers of openKONSEQUENZ modules according to architecture related issues (this document).
ACQC	Architecture Committee and Quality Committee	Architekturkomitee und Qualitätskommite	AC and QC together
	Core Module	Core Module	An oK module that provides cross cutting services that are not special to the energy domain but providing services for multiple user modules or domain modules (see chapter “Solution Strategy; oK Multilayer Architecture”).
	User Module	Fachliches Modul	An oK-application, a user from a DSO uses for solving his/her use case.

DSO	Distribution System Operator	Verteilnetzbetreiber (VNB)	Manages the distribution network for energy, gas or water
ESB	Enterprise Service Bus		Central instance for exchange of data to overcome point-to-point connections
IaaS	Infrastructure as a Service		Cloud Service for hosting.
IP	Intellectual Property	Geistiges Eigentum	Protections for copyright, patents,..
	Domain Module	Domain Modul	An oK module that provides domain specific services for multiple user modules (see chapter solution strategy; oK Multilayer Architecture)
oK	openKONSEQUENZ	openKONSEQUENZ	Name of the consortium of DSOs
QA	Quality Assurance	Qualitätskontrolle	Check, if solutions fulfilling requirements to quality
QC	Quality Committee	Qualitätskomitee	Gives framework and constraints according to quality for oK projects
QC handbook	Quality Committee handbook	Handbuch des Qualitätskommitees	Textural guideline for module developers of openKONSEQUENZ modules according to quality related issues.

SCADA	Supervisory Control and Data Acquisition	Netzleitsystem	System, that allows DSOs view/control actual parameters of their power grid.
VPN	Virtual Private Network	Virtual Private Network	Extends private networks across a public network.

TODO for architecture documentation
by module developer according to Chap. 12:

- Document important or misleading abbreviations and terms