Technical Development Requirements

for

Federation Architecture for Composed Infrastructure Services (FACIS)
Multi-Party Service Level Agreement Demonstrator

FACIS.SLA.Demo

Table of Contents

Open-Source Software....3
AI-Assisted Development....4
Runtime Environments....4
Programming Languages....5
Documentation....5
QA Requirements....7
DevOps Tools....8
Architecture Decision Records (ADR)....9

Open-Source Software

- Mandatory use of XFSC ORCE[1] for
  - Service orchestration and API usage to boundary systems
  - Usage of the FACIS UI Builder
- Deployment on Kubernetes
  - Deployment into the Kubernetes cloud environment provided by FACIS (IONOS and T-Systems Open Sovereign Cloud).
- Mandatory use of Kubernetes and Helm for
  - Automated deployment
  - Update and uninstall processes
  - Resource and ingress management
- Helm charts
  - Deliver Helm charts with each release
- Mandatory integration with XFSC OCM W-Stack and PCM Cloud for
  - Credential issuance, presentation verification and status list via the OCM W-Stack, consumed as provided by XFSC FAP PCI
  - Credential holders in PCM Cloud
- Mandatory use of the FACIS UI Builder package for
  - Visual frontends
- Mandatory use of XFSC DCS for
  - All preparation, negotiation, contracting, approval and signing workflows
- Repository rules
  - All code, Node.js logic, UI schemas, Helm charts and scripts must reside in the designated FACIS repository, that will be communicated after the award of the tender
  - CI runs on GitHub Actions with security checks
  - No parallel private repositories allowed
  - Daily commits are mandatory: the individual code must be pushed to a feature branch in the Client's software repository at the end of each day of development.
  - Changes reach the `develop` and `main` branches only through pull requests. Branch protection requires passing CI and at least one approving review by a person who holds committer status in the Eclipse XFSC project or is explicitly designated by the Client (via CODEOWNERS). Approvals by AI tools or bot accounts do not count.
  - Pull requests shall be small enough to be reviewed in one sitting; the reviewer may request a split.
  - All code and charts live in the client-designated repository.
- Compliance with Apache License, Version 2.0:
  - Text version: https://www.apache.org/licenses/LICENSE-2.0.txt
  - SPDX short identifier: Apache-2.0
  - OSI Approved License: https://opensource.org/licenses/Apache-2.0
- Only FOSS code must be used and must be compatible with Apache License 2.0
  - The project should consume third-party content under one of the approved licenses listed here: https://www.eclipse.org/legal/licenses.php#approved. The client must be informed in written form when exceptions are planned to be included.

[1] https://github.com/eclipse-xfsc/orchestration-engine

    - The client must confirm any existing FOSS code that is planned to be included in the deliverables.
    - The client must confirm API and FOSS component versions after contractor's in-depth assessment
    - This applies equally to dependencies proposed by AI tools (see AI-Assisted Development).
- No redundant functions in the XFSC repository.
- A project git repository will be made available, and all contractors are obliged to use this repository as master on a daily base. A parallel contractor repository is not allowed.
- All work must be done in compliance with Eclipse Contributor Agreement 3.1.0 or higher: https://www.eclipse.org/legal/ECA.php

AI-Assisted Development

AI coding assistants may be used under the Generative AI Usage Guidelines for Eclipse Committers (https://www.eclipse.org/projects/guidelines/genai/) and ADR 007. The following rules apply in addition.

- Accountability
    - Every commit is authored by a named human contributor with a valid ECA, who is fully responsible for its content, including correctness, security and IP due diligence.
    - AI agents do not commit, push or merge under their own identity.
    - AI-generated content goes through the same review, testing and licence checks as any other contribution.
- Disclosure
    - Where a file contains substantial AI-generated content, a comment directly below the SPDX/licence header names the tool, e.g. `// Portions generated with <tool>, reviewed by <name>`.
    - The pull request template contains a checkbox “AI tools used: yes/no; if yes, which”.
    - Trivial assistance (autocomplete, renaming, typo fixes) needs no disclosure.
- Confidentiality
    - Client credentials, test credentials, Kubernetes secrets and non-public client documents must not be entered into AI tools.
    - Only AI tools whose terms of use exclude training on user inputs may be used on project code.
- Dependencies and public code
    - Dependencies proposed by AI tools are subject to the same Eclipse Dash licence check and client confirmation as any other FOSS component.
    - Where an AI tool offers a filter against reproducing public code, it must be enabled.
- Tool list
    - The AI tools intended for use are named in the project plan under “DevOps management and tools” and confirmed with the plan approval.

Runtime Environments

- Kubernetes v1.29 or higher in the FACIS cloud environment.
- Helm for deployments
- ORCE runtime for all SLA.Demo components
- No Docker Compose permitted

Programming Languages

- Node.js (JavaScript/TypeScript) for the SLA.Demo component logic inside ORCE;
- Bash for automation scripts under /scripts
- YAML for Helm templates and Kubernetes manifests
- JSON-LD for templates
- No further service languages: all developed SLA.Demo components run as ORCE instances

Documentation

- All documentation must follow Eclipse Project Handbook guidelines: https://www.eclipse.org/projects/handbook/
- Deliverables:
  - Deployment and teardown instructions
  - Helm chart documentation
  - Demonstration script and runbook covering demonstration scenario from SRS Section 5
  - Troubleshooting guide
  - CONTRIBUTING.md section “Use of AI tools” and pull request template with AI disclosure checkbox (see Section AI-Assisted Development)
- All documentation must be included in docs/ or README.md in the project repository.
- Project repository:
  - High-level description
  - Packaging and container
  - Deployment and preconditions
  - Changes and additions to source specifications
  - Functional breakdown for orchestrated flows
  - Licenses
  - OSS dependencies and external OSS references

General Principles

- GitHub uses a monorepo-style approach, where multiple sub-projects are organized in folders within a single repository. Each repository has one-version history, so nested independent Git projects are not supported by default.
- Single Source of Truth: Each repository is the authoritative source for its respective development activity.
- Unified Content: All relevant concepts, specifications, and implementation materials must reside in the same repository.

Repository Creation & Contributor Onboarding

- Repository Creation: Eclipse XFSC Project Leads are responsible for creating the main project repositories and applying the correct naming conventions. Communication with Eclipse XFSC Project Leads via the Client.
- Contributor Onboarding: The implementation team must complete the Contributor Onboarding (https://eclipse-xfsc.github.io/landingpage/developer-guide/on-boarding) to be onboarded to the XFSC Eclipse Project.

- Contributor onboarding grants contributor rights only. Merge approvals require committer status in the XFSC project or designation by the Client (see Repository rules); the project plan names the approving reviewers and their availability.

Documentation Guide

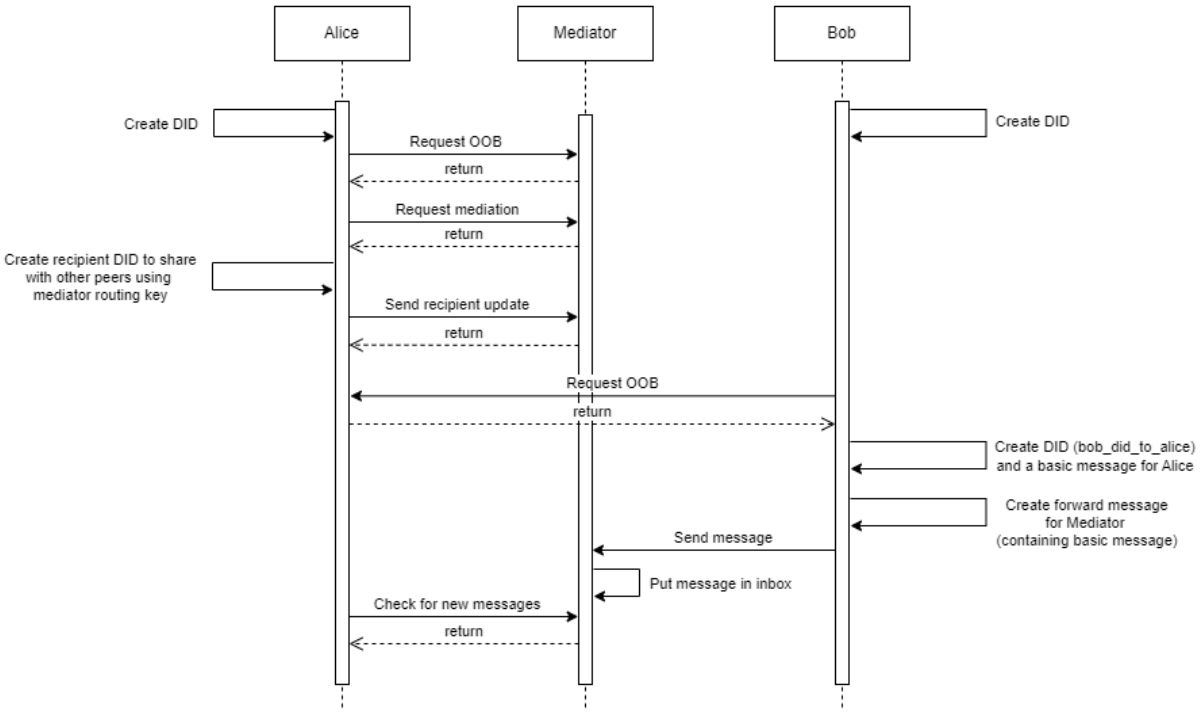
- Generator: readthedocs, mkdocs
- Product Documentation: AsciiDoc, reStructuredText (RST)
- Architecture Modelling: ArchiMate, UML, https://www.drawio.com/
- Interface description with versioned schemas, API definitions and examples for all REST endpoints; Node-RED node specs for the ORCE custom nodes.
- The final repository will be communicated after the award of the tender.

README.md

This markdown file should include the

- Project Overview (brief description of the SLA.Demo component and its ORCE flows),
- Installation & setup (Dependencies, environment variables and configuration),
- Technical documentation Links,
- use case diagram (if applicable to understand the service functionality).

The example use case diagram (basic DIDComm v2 flow between two peers via a mediator) remains as in the current version, referenced from https://gitlab.eclipse.org/eclipse/xfsc/common-services/didcomm-connector:



Example project directory structure:

```
Repo/
|-- README.md
|-- CONTRIBUTING.md
|-- docs/
|   |-- features.md
|   |-- deployment.md
|   |-- api.md
```

The docs/ directory should contain detailed documentation to help developers, contributors and users to understand the project effectively.

A template for the CONTRIBUTING.md: https://github.com/auth0/open-source-template/blob/master/GENERAL-CONTRIBUTING.md

OSS Microservice Repository

File/Folder	Purpose	Status (✓/✗)
.github/workflows	The Eclipse Dash license scanners and build workflows must be referenced in the repo from the standard org: https://github.com/eclipse-xfsc/dev-ops/tree/main/.github/workflows (example: https://github.com/eclipse-xfsc/oid4-vci-vp-library/tree/main/.github). Includes the pull request template with AI disclosure checkbox.	
deployment/helm, deployment/docker	Folders for Helm deployments and Docker image builds.	
README.md	Project overview, installation, setup, architecture, links to documentation folder, contributors responsible for contact info/e-mail address, concept and scope	
CONTRIBUTING.md	Guidelines for contributors, process release and issue reporting. Section „Use of AI tools“.	
CODEOWNERS	Approving reviewers (XFSC committers or Client-designated) for branch protection	
LICENSE	License terms for the project	
CODE_OF_CONDUCT.md	Rules and behavior expected for the community	
docs/	Contains detailed documentation about the project / specifications or models. This doc section should be hosted on GitHub Pages.	
docs/ci-cd.md	CI/CD pipeline configuration, GitHub Actions and deployment.	
docs/api-docs.md	API documentation (Swagger, OpenAPI)	

QA Requirements

ORCE Automation QA

- Deploy, redeploy and uninstall must run with zero manual intervention
- Logs must use structured JSON
- Error output must be machine-readable and exposed via ORCE context

Runtime Validation

- TLS 1.3 enforced for all endpoints
- Secrets stored in Kubernetes Secrets; no plaintext in logs
- Required Kubernetes objects created correctly in the client-provided namespace: ingress, service, deployment
- External dependencies (OCM W-Stack, PCM Cloud, Federated Catalogue, DCS, FAP DCM, FAP PCI, trust endpoints, Trusted Cloud service listing) reachable

Code Readability and Maintainability

- Code passes ESLint and Prettier with the repository configuration in CI.
- Exported functions and ORCE custom nodes carry JSDoc.
- No minified, obfuscated or unexplained generated code is committed; generated artefacts such as schemas are produced by a documented script.
- By approving a pull request, the reviewer confirms that they understand the change and consider it maintainable by the project community. This applies regardless of whether AI tools were used.

Test Suite

Must include the following tests:

- Successful deployment
- Invalid parameters
- Idempotent redeployment
- Uninstall
- Integration tests run once in the FACIS cloud environment

Expected results in tests are taken from the acceptance criteria, not derived from the implementation. This applies equally to AI-generated tests and is checked in review.

Test Reports

- Deliver the CI test report with each tagged release

Target hosting environments for QA

- Reference cluster: FACIS cloud Kubernetes environment
- Helm charts must pass lint and dry-run checks as part of QA.

DevOps Tools

- Git
- GitFlow
- GitHub Actions (CI/CD)
- Release artifacts: Helm repo, OCI images, CI test reports.

Architecture Decision Records (ADR)

ADR 001 – Deployment Engine

Decision: Helm is the mandatory deployment engine for all SLA.Demo components.

Consequence: Repeatable Kubernetes-native deployments.

ADR 002 – Orchestration Layer

Decision: ORCE is the mandatory runtime of all SLA.Demo components.

Consequence: One runtime for all components; logic in unit-testable modules behind thin custom nodes

ADR 003 – Automation Stack

Decision: Node.js + Bash + YAML for backend logic and automation; UI built with the FACIS UI Builder package.

Consequence: Stable and consistent execution environment.

ADR 004 – Identity Integration

Decision: Identity with DID-based verifiable credentials issued via the OCM W-Stack and held in PCM Cloud

Consequence: Consistent identity model across all SLA.Demo components

ADR 005 – Security Baseline

Decision: TLS 1.3, RBAC, Kubernetes Secrets for service secrets (verifiable credentials remain in PCM Cloud,), log masking.

Consequence: Secure, auditable deployment operations.

ADR 006 – Logging and Observability

Decision: Structured JSON logs exposed via ORCE context outputs.

Consequence: Machine-readable logs for testing and monitoring.

ADR 007 – AI-Assisted Development

Decision: AI coding assistants may be used under the Eclipse Foundation Generative AI Usage Guidelines for Eclipse Committers and the section AI-Assisted Development. Every commit is authored by a named human contributor with a valid ECA, who is fully responsible for its content. AI agents do not commit, push or merge under their own identity, and AI review output never counts as an approval.

Consequence: AI-assisted productivity without diluting accountability, IP due diligence or Eclipse governance; all merges remain approved by humans entitled to do so.