OpenSCD next architecture

This is in DRAFT. Feedback or improvements are welcome using the comment feature in confluence.

To solve some limitation with the OpenSCD monolith architecture; and improved architecture is created. The Motivation for creating a new generation of OpenSCD-next architecture:

1. Make it more easy to build your own distribution
2. Make it more easy to create custom plug-ins
3. Less tangled code
4. Reusable code
5. Faster development speed (once it is ready)
6. Plug-ins can be build in any code language and framework (as long as they adhere to the specifications)

Practical example of the current limitations

Practical example for CoMPAS. CoMPAS currently relies on "fork" of OpenSCD in order to add the needed functionality:

com-pas/compas-open-scd: A substation configuration description editor for projects using SCL IEC 61850-6 Edition 2 or greater (github.com)

The goal for CoMPAS would be to add the CoMPAS specific code as add-on to OpenSCD-next in form of plug-ins. The current "fork" will no longer be needed. CoMPAS will just make a distribution of OpenSCD including the CoMPAS plugins/extensions.

OpenSCD next product vision

Next Generation

OpenSCD NEXT is the next generation version of OpenSCD aimed at flexibility and extensibility. NEXT features a modular structure that enables third parties to develop on top of OpenSCD CORE, thus enabling the community to add plugins, add-ons and components on top of OpenSCD CORE, while core functionality can be maintained separately.

Community focused

OpenSCD NEXT has a focus on community, and is essentially a tool for collaboration, in an environment supporting the energy transition. It enables third parties to easily participate.

Marketplace

OpenSCD NEXT features a bazaar model, where providers can develop functionality together and share what they have developed in a generic way for substation configuration.

Create your own packages

OpenSCD NEXT enables providers to dynamically create packages for their own purposes or packages that provide generic value for the community.

Architecture principles for OpenSCD-core

Some architecture principles are written in order to guide the architecture and implementation.

1. Use the standard webAPI as much as possible (developer.mozilla.org)
2. Interaction with the SCL should be easy
    a. Libaries for specfic taks
    b. Do and undo functionality
3. Integrate small functionality within wider platform quickly (easy to deploy)
4. Clear and documented API's for plug-in authors
5. Plug-in authors can use components to speedup the development and give a consistent look and feel (optional)

Technical implications:

Modular and loosely coupled

Limit dependencies

General functional aspects of OpenSCD next

- Embed OpenSCD next as embedded application is not foreseen yet. It might be possible with an iframe. The current technology might already allow it.
- Support multiple OpenSCD instances is not supported
- OpenSCD provides: front-end for graphical interaction

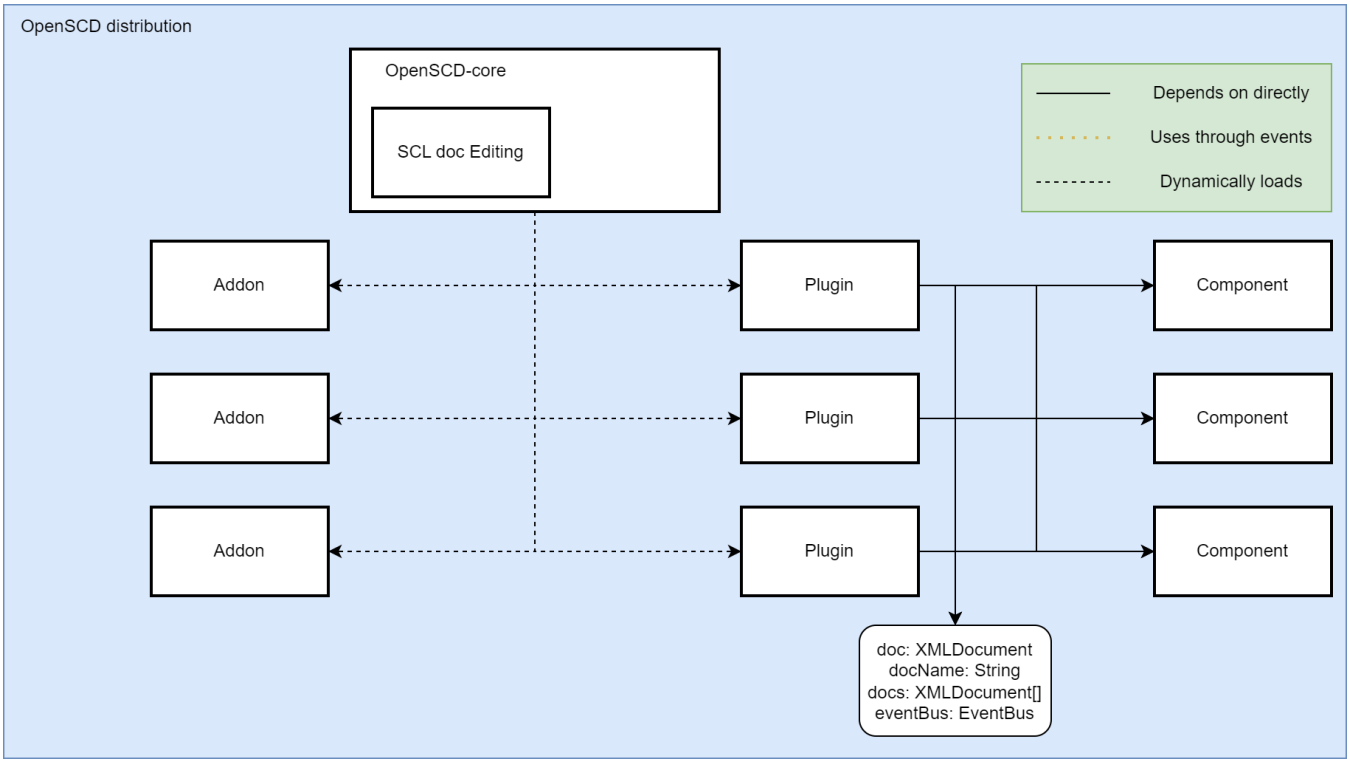
Solution: OpenSCD-core architecture overview

The OpenSCD-next architecture consist of 4 major software components:

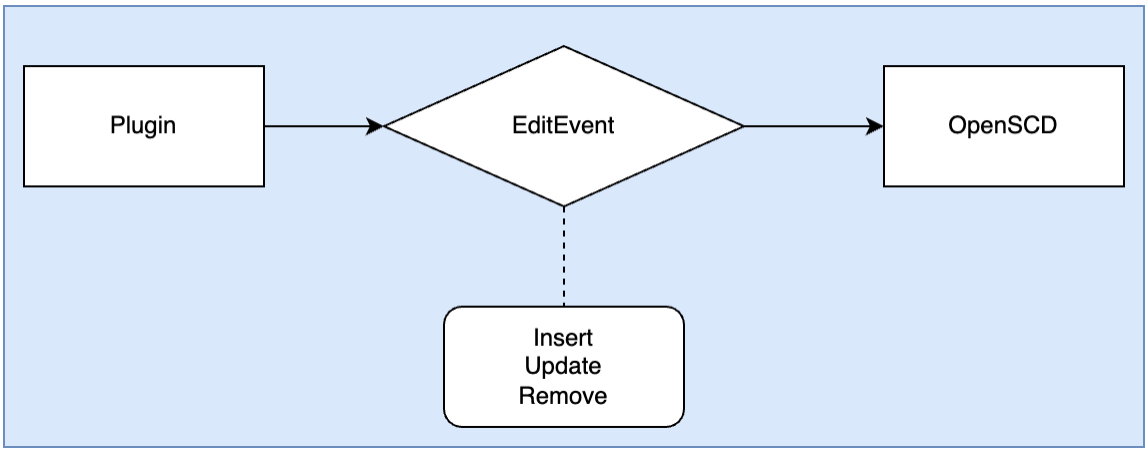
- OpenSCD-core
- Plug-ins
- Components
- Add-ons

Overview of the global architecture

Components and their interaction. OpenSCD-core loads all the "components/plugins etc".



A plug-in can use EditEvent to make modifications in OpenSCD-core.



OpenSCD-Core

OpenSCD-core is the main application of OpenSCD-next. OpenSCD-core will be loaded once people open the application. OpenSCD-core sets the requirements and API's for the plug-ins.

Functionality should should be stable and long lasting in order to have stable ecosystem.

Selection criteria for incorporate stuff into OpenSCD-core:

- Re-usable for many plug-in authors
- Should be maintainable in the long run in order to have a stable ecosystem.

OpenSCD-core consists of the `open-scd` WebComponent, which is responsible for loading plugins and addons.

The `open-scd` WebComponent is the outer shell of OpenSCD, containing limited functionality in order to make it flexible.

Editing of a (SCL) Document *should* be handled by OpenSCD-core. OpenSCD-core provides an API for document editing by listening to CustomEvents. These CustomEvents can be dispatched by plug-ins.

Based on a configuration file OpenSCD-Core loads the different plug-ins/addons etc.

Functionality of OpenSCD-core:

Loading addons/plug-ins

Display the selected plug-in

Handling SCL edits

Host the SCL doc editing

SCL doc editing

The SCL doc editing is reponsible for manipulating the SCL. It takes care of the right sequence/order of SCL edits. For more information see OpenSCD-core handling SCL Edits.

OpenSCD Plug-ins

An OpenSCD plug-in is an addition to OpenSCD-Core to add functionality. This could be generic 61850 functionality or vendor/utility specific functionality. Examples: Datatemplate editor, substation-section editor, GOOSE editing, validation etc.

OpenSCD-Core supports 2 types of plug-ins: `menu plug-ins` and `editor plug-ins`. These plug-ins have a different behavior/requirements.

For more information see: OpenSCD-core plug-in

OpenSCD Components

OpenSCD Components are reusable WebComponents. Components *can* be used by plug-ins to build functionality faster, plug-in authors can also decide to build everything themselves.

Filtered-lists

Examples: filtered-lists are used in many plug-ins. In order to re-use the code, a filtered-list component is build and published on NPM.

OpenSCD Addons

OpenSCD addons are an extension of OpenSCD-Core to split functionality. This functionality can be pure technical and requires no custom UI. Examples for some OpenSCD add ons are Wizarding, Validating, Waiting, etc.

OpenSCD addons will be a replacement for current mixins, so there is no/less need to `fork` OpenSCD-Core and to build mixins on top of the fork. Addons allow experiments with (potential) new core functionality. Successful addons can be merged into OpenSCD-core.

Wizarding

Example: Wizarding API is hard to make perfect in the first run. If started with an addon for wizarding, we can experiment/use it. If it turns out that the addon is missing crucial functionality. We can introduce a new add-on next to the old addon (with its limitations).

Solution: There will be 2 POC's by Tamás Russ and pascal wilbrink . Comparing those will give the best solution.

For more information about mixins: TypeScript: Documentation - Mixins (typescriptlang.org)