Development of a web-based application for the degradation configuration of IoT components

Project Report

Steffen Knoblauch steffen.knoblauch@uni-ulm.de

July 9, 2021

Contents

1 Introduction

More and more IoT (Internet of Things) devices are finding their way into our daily lives, making it possible to combine a wide variety of end devices or sensors and design new composite systems. Such systems can encounter internal problems, for example, when a component no longer operates properly. The system as such must react to this and, for example, cease functionality or operate in another way.

This is where the SORRIR project comes in to develop a self-organising, resilient execution platform for IoT services. In order to make such a system resilient, i.e. resistant to errors of the subcomponents of the system, different levels were introduced in which such a system can operate.

A system that is composed of several subcomponents may be forced to change the level if, for example, one of these subcomponents is no longer functional. In the worst case, up to the complete deactivation of the system. In order to prevent a system from going directly into the deactivated state whenever an error occurs, various intermediate levels, the so-called degradation levels, can be defined. For example, a degradation level can continue to provide some of the original functionality, but with some limitations depending on the non-functional subcomponent.

An example of this is a smart barrier in a parking garage, which in the ideal case, i.e. when all subcomponents are functioning, recognizes a car by means of a camera and is opened automatically. If, for some reason, the camera failed, the system would no longer be able to function. However, if one uses the degradation levels, a part of the functionality could be guaranteed. For example, by opening the barrier manually with an ID card. To configure such degradation levels a json file is required which defines the different transitions between the levels. For example, what should happen when a certain subcomponent is no longer functional.

This is the starting point of this project, in which the goal is to create a graphical interface in form of a web application that allows to configure a system with the different degradation levels and finally to export it as Json. In order to change or create a configuration, several components for the web application are required, like the definition of subcomponents or the import of an existing configuration.

This report starts with a short description of the technology stack, in section 2, that was used to implement the web interface. It is followed by the architecture of the application, in section3, that includes a description of the general components, their usage, and some additional information about the implementation. The last section is the conclusion that contains a short summary as well as potential further enhancements.

2 Technology Stack

- React (create-react-app)
- Typescript
- most important libraries:
  - material-ui (Material design of the application)
  - react-syntax-highlighter (SyntaxHighlighter: preview of the import and the current configuration in the export)
  - ajv (Json Validation for the import)
  - react-dnd: Drag and Drop Library for react

3 Architecture

TODO: Use figures for the different sections (e.g. import successful and unsuccessful)

3.1 General concepts

- Modular
- Extensible (integration of other steps)

3.2 Wizard

- Component to allow a step by step configuration
- Different configurations - with and without import in the app start possible
- Restart of the wizard in every step possible (will delete all changes and return to the start)
- Easy extensible: add new steps - internally called views by:
  - Extending the View- enum with the new view ("src\util\Views.ts")
  - Providing a Label (will be used as caption) ("src\util\ViewLabelResolver.ts")
  - Adding the View (react component) that will be displayed for the step in the Wizard ("src\components\wizard\ViewSelector\ViewSelector.tsx")

3.3 Import

- Step to import an existing configuration (json file)
- Will validate the Json file and display errors (validates if it can be parsed and will also display if it matches the formal description of the configuration (details in the validation section)
- After a successful import - internal configuration state will be updated and the current state will be displayed in a json viewer (with syntax highlight)
- internal configuration is based on a several interfaces found in "src\models"

3.4 Validation

- Validates if the file can be parsed as Json element
- If it can be parsed the validation will be done
- short explanation of the different schema types?
- use of ajv (support of JSON Schema and JSON Type Definition)

- atm most tools support Json Schema (cite `https://ajv.js.org/guide/schema-language.html`) → selection for our usecase - but no offical RFC only internet draft
- ajv used because it supports both - migration later possible (to RFC8927 JSON Type Definition)
- Short overview of the implemented schema for the configuration (perhaps using UML diagram)

3.5 Subcomponent configuration

- Subcomponents required for the creation of degradation levels
- Subcomponent has an id, a name and several shadowmodes (internal state of the subcomponent)
- each shadowmode has an id and a name
- In order to be able to manage the relevant subcomponents for the component, a table with a creation/edit and deletion dialog is offered
- The creation/edit dialog offers several functions:
  - Internal validation function (non empty fields + check for already existing subcomponents with the same id)
  - id autofill (based on the name) if no value is set

3.6 Degradation Level Management

- Creation/Deletion in the Degradation and Upgrade Configuration views
- Create/Edit with a Dialog:
  - name and id as string
  - the subcomponent states of the level (will show all possible values of a subcomponent and to choose what has to be true for this level)
  - creation of internal states of the Level (with custom chip input)
  - validation if id exists/ etc.
- Multi selection in the Degradation and Upgrade Configuration View

3.7 Degradation and Upgrade configuration

- Custom Graph View for the Hierarchy (Recursive generation of the Graph)
- Drag and Drop Support to update the Graph (degradation level hierarchy)
- Off state as default node

- Explain model on how to save the hierarchy (LevelChange model)
- shows all levels even the ones that are not inserted in the hierarchy yet
- Upgrade: Inverse of the Degradation Graph - same functionality as Degradation Graph
- Both will be saved separately in the configuration and are also separate steps in the wizard

3.8 LevelChange configuration

- configure the state in which the level after the degradation or upgrade is
- separate step in the wizard for both degradation and uprade
- shows for all LevelChanges the states of the corresponding level and allows the selection of the state the level will be in after the degradation/upgrade

4 Conclusion