IMPaCT:  
“Configuration and Service Toolkit Prototype”

Deliverable AD0039

|  |  |
| --- | --- |
| Document type | Deliverable |
| Author | TNO/Henk Ensing,  KIT/Jan Müller,  FZI/Ingo Mauser,  TNO/Wilco Wijbrandi,  TNO/Johan van der Geest |
| Version | 1.0 |
| Creation date | December 19, 2014 |
| Number of pages | 30 |

**For internal use only - Restricted to EIT ICT Labs**

Index

[IMPaCT: “Configuration and Service Toolkit Prototype” 1](#_Toc406663729)

[1 Introduction and Overview 5](#_Toc406663730)

[1.1 Introduction 5](#_Toc406663731)

[1.2 Decision Process 5](#_Toc406663732)

[1.2.1 Idea: OSGi based Software Architecture 5](#_Toc406663733)

[1.2.2 Evaluated Third-party Software Components 7](#_Toc406663734)

[1.2.2.1 JointJS 7](#_Toc406663735)

[1.2.2.2 jsPlumb 8](#_Toc406663736)

[1.2.2.3 NoFlo 8](#_Toc406663737)

[1.2.2.4 Eclipse GEF (Graphical Editing Framework) 9](#_Toc406663738)

[1.2.3 Decision: Usage of JointJS, Zurb Foundation, jQuery, Jackson, and Jetty 10](#_Toc406663739)

[2 Toolkit Prototype 11](#_Toc406663740)

[2.1 Overview 11](#_Toc406663741)

[2.2 Configuration Data Model 12](#_Toc406663742)

[2.2.1 Aspects of the Configuration Model 12](#_Toc406663743)

[2.2.2 Configuration Model Architecture 12](#_Toc406663744)

[2.2.2.1 Model Entry Points 13](#_Toc406663745)

[2.2.2.2 ModelElement 14](#_Toc406663746)

[2.2.2.3 Description 14](#_Toc406663747)

[2.2.2.4 Property 14](#_Toc406663748)

[2.2.2.5 ArtifactId 14](#_Toc406663749)

[2.2.2.6 Component 14](#_Toc406663750)

[2.2.2.7 Interface 14](#_Toc406663751)

[2.2.2.8 Port 14](#_Toc406663752)

[2.2.2.9 Input 14](#_Toc406663753)

[2.2.2.10 Output 15](#_Toc406663754)

[2.2.2.11 Project 15](#_Toc406663755)

[2.2.2.12 Application 15](#_Toc406663756)

[2.2.2.13 Instance 15](#_Toc406663757)

[2.2.2.14 Wire 15](#_Toc406663758)

[2.2.2.15 Source 15](#_Toc406663759)

[2.2.2.16 Target 15](#_Toc406663760)

[2.2.2.17 Constant 15](#_Toc406663761)

[2.2.2.18 Bundle 15](#_Toc406663762)

[2.2.2.19 Compound 16](#_Toc406663763)

[2.2.2.20 ProxyInputWire 16](#_Toc406663764)

[2.2.2.21 ProxyOutputWire 16](#_Toc406663765)

[2.3 Architecture 16](#_Toc406663766)

[2.4 Software Components 17](#_Toc406663767)

[2.4.1 Third-party Components 17](#_Toc406663768)

[2.4.2 Developed Components 17](#_Toc406663769)

[2.4.2.1 JointJS shapes 17](#_Toc406663770)

[2.4.2.2 Palette 18](#_Toc406663771)

[2.4.2.3 Inspector 19](#_Toc406663772)

[3 Evaluation 20](#_Toc406663773)

[3.1 Evaluation: Charging Station 20](#_Toc406663774)

[3.1.1 Configuration of IMPaCT charging spot controller 21](#_Toc406663775)

[3.1.1.1 Frontend project 21](#_Toc406663776)

[3.1.1.2 Backend processing 24](#_Toc406663777)

[3.1.2 Parameters configured during evaluation 25](#_Toc406663778)

[3.1.3 Test execution 25](#_Toc406663779)

[3.2 Evaluation: Smart Building 27](#_Toc406663780)

[4 Conclusions and Outlook 30](#_Toc406663781)

Figures

[Figure 1: Initial OSGi-based Solution proposed by TNO 6](#_Toc406742658)

[Figure 2: Prototype for the IMPaCT Toolkit developed with JointJS 7](#_Toc406742659)

[Figure 3: Prototype of the IMPaCT Toolkit build with jsPlumb 8](#_Toc406742660)

[Figure 4: Prototype of the IMPaCT Toolkit build with NoFlo 9](#_Toc406742661)

[Figure 5: Example of an Eclipse GEF application 9](#_Toc406742662)

[Figure 6: Deployment diagram of the IMPaCT Toolkit 11](#_Toc406742663)

[Figure 7: EMF meta-model for the IMPaCT Toolkit 13](#_Toc406742664)

[Figure 8: Architectural overview 16](#_Toc406742665)

[Figure 9: The three custom shapes that were developed 17](#_Toc406742666)

[Figure 10: The palette for the IMPaCT Toolkit frontend 18](#_Toc406742667)

[Figure 11: The dialog that is shown when adding a component 18](#_Toc406742668)

[Figure 12: The inspector for the IMPaCT Toolkit frontend 19](#_Toc406742669)

[Figure 13: Tested scenario with the Charing Station 21](#_Toc406742670)

[Figure 14: The root application for the charging station project 22](#_Toc406742671)

[Figure 15: The implementation of the OPERATORCONFIG compound 23](#_Toc406742672)

[Figure 16: The implementation of the CCBoxInternal compound 24](#_Toc406742673)

[Figure 17: On the left the frontend, on the right the backend 25](#_Toc406742674)

[Figure 18: The initial voltage was set to 250 Volts 26](#_Toc406742675)

[Figure 19: The updated voltage after changing it in the IMPaCT Toolkit frontend 26](#_Toc406742676)

[Figure 20: KIT Energy Smart Home Lab 27](#_Toc406742677)

[Figure 21: Setup of FZI House of Living Labs 28](#_Toc406742678)

[Figure 22: IMPaCT Toolkit configuration of “KIT Energy Smart Home Lab” reference design 28](#_Toc406742679)

[Figure 23: IMPaCT Toolkit: Configuration of the Smart Building “FZI House of Living Labs” 29](#_Toc406742680)

# Introduction and Overview

## Introduction

The EIT ICT Labs – IMPaCT project was started in 2014 with a specific innovation objective in mind:

* To create market readypower controllers for electric vehicles
* prepare those for smart distribution grids, and
* virtual power plants

The main goal of activities is to achieve an easy configuration during integration, and to offer services to support this time critical project phase.

This particular document is the deliverable AD0039 for the third activity (A1403) in the project and is a description of the “Configuration and Service Toolkit Prototype”. In this task, we have described the A1403 Technology:

* Experimentation
* Charge Spot Scenarios - Prototyping and Testing

At this stage in the project, we will focus solely on the application of the IMPaCT controller in the area of EV charging and more specifically on the usage within a charge spot. For 2015, it is planned to look beyond this initial scope and include the building environment domain as an additional application area for the IMPaCT controller.

In this document, the implementation of the IMPaCT Toolkit prototype is documented. Additionally, the prototyping and finally the testing and evaluation of the IMPaCT Controller with the IMPaCT Toolkit in the Charging Station and Smart Building Scenarios are addressed and documented.

## Decision Process

Before implementing the IMPaCT Toolkit, several possible solutions have been discussed. One of these solutions would have been based on OSGi as outlined in the next subchapter. Within these discussions, TNO described the advantages and possible features of an IMPaCT Toolkit that uses OSGi. Additionally, OSGi as a whole was explained to the project partners by TNO.

After deciding for a JavaScript-based frontend with a Java-based backend (see also Figure 8), several third-party software components have been evaluated for further possible usage within the IMPaCT Toolkit. Finally, it was decided that the best solution would be to implement the tool as a stand-alone application with a Java back-end and a JavaScript front-end that can be used with a web browser (but does not require an Internet connection).

### Idea: OSGi based Software Architecture

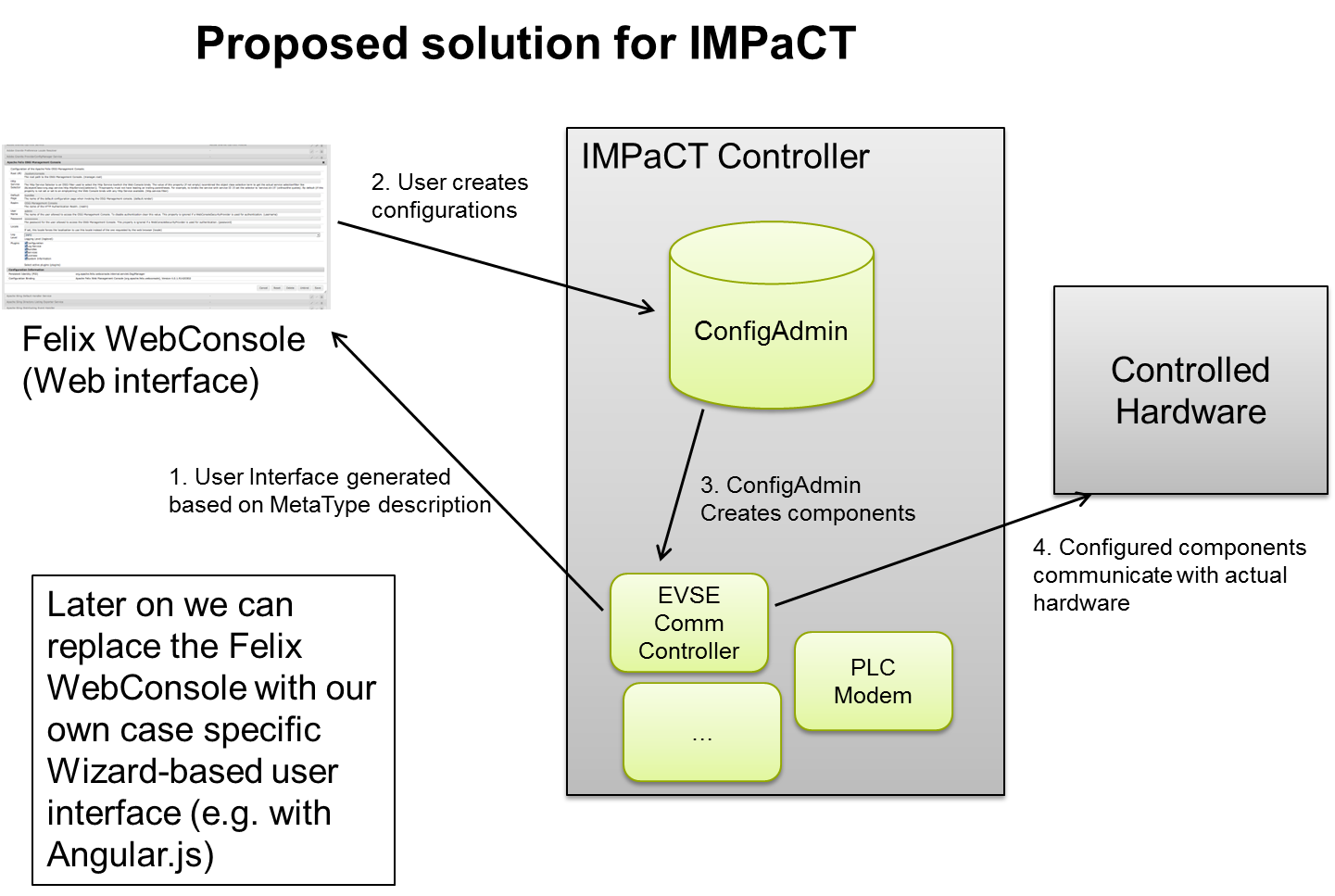


Figure 1: Initial OSGi-based Solution proposed by TNO

OSGi (Open Service Gateway initiative) is a specification for an open service platform for the Java programming language, with several closed-source and open-source implementation. Together with the specification of the platform, there are specifications for common services, which include *Configuration Admin* (or *ConfigAdmin* for short) and *MetaType*.

The goal of the IMPaCT Configuration and Service Toolkit is to configure software component on the IMPaCT Controller, which are not known by forehand. This requires a generic way of describing configuration options and possible configuration instances, as well as having a way of storing the configuration instances.

MetaType is a service, which can be used to describe the possible configurations for software components. ConfigAdmin is configuration database. This seems like a good fit for the implementation of the IMPaCT Toolkit. By reusing these services, it would also be possible to make use of existing user interfaces interfacing with these components, such as the Felix WebConsole.

However, there is a mismatch between OSGi and the IMPaCT Toolkit. With MetaType and ConfigAdmin the goal is to configure software components running on the OSGi execution environment, where in our case the execution environment is the IMPaCT Controller. In addition, our focus is more on wiring software components, where with ConfigAdmin the wiring requires some extra software components. Because of these reasons, it was decided not to use the OSGi based services for the implementation, but to do use the principles and to use it for inspiration.

### Evaluated Third-party Software Components

Before implementing the actual IMPaCT Toolkit prototype, several third-party software components have been evaluated.

#### JointJS

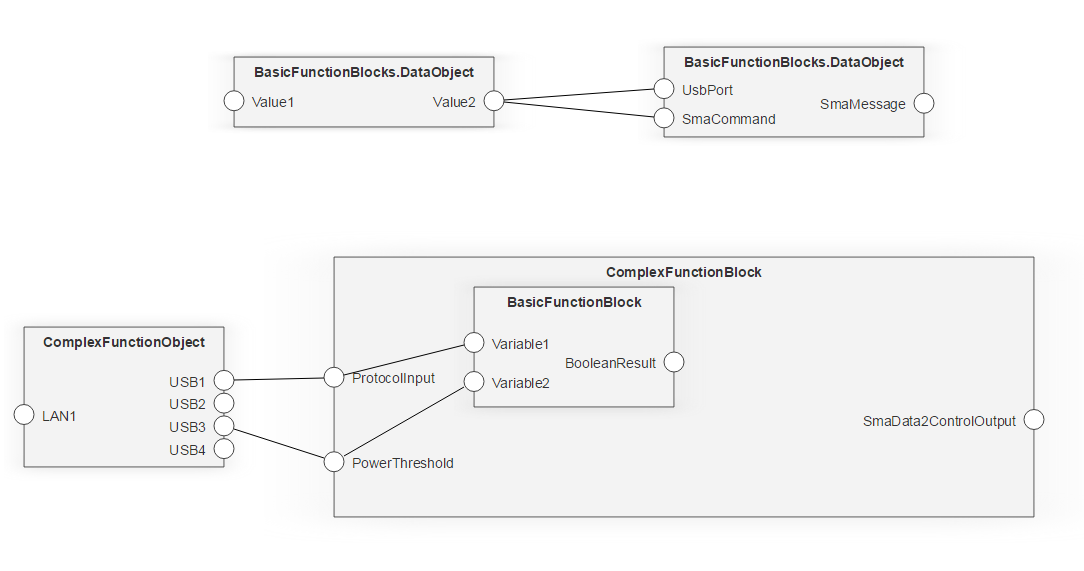


Figure 2: Prototype for the IMPaCT Toolkit developed with JointJS

JointJS is an open source (Mozilla Public License Version 2.0) HTML5 software library for developing modeling tools that can run inside a web browser (see Figure 2). The library was developed with the Model-View-Controller pattern in mind, and keeps track of the underlying model of the graph. This makes it easier to build interactive editors. The project has a well-documented API and provides several tutorials .

There is also a commercial software library for developing editors for JointJS available, called Rappid. It offers user interface functionalities such as undo and repeat, copy and paste, editing the properties of shapes and wires using an inspector and creating new shapes from a palette.

JointJS was chosen as the visualization library for the IMPaCT Toolkit. We decided to not use the commercial Rappid toolkit in this face of the project, as the effort to create the extra functionalities ourselves was low. It also gave us more control over the user interface.

#### jsPlumb

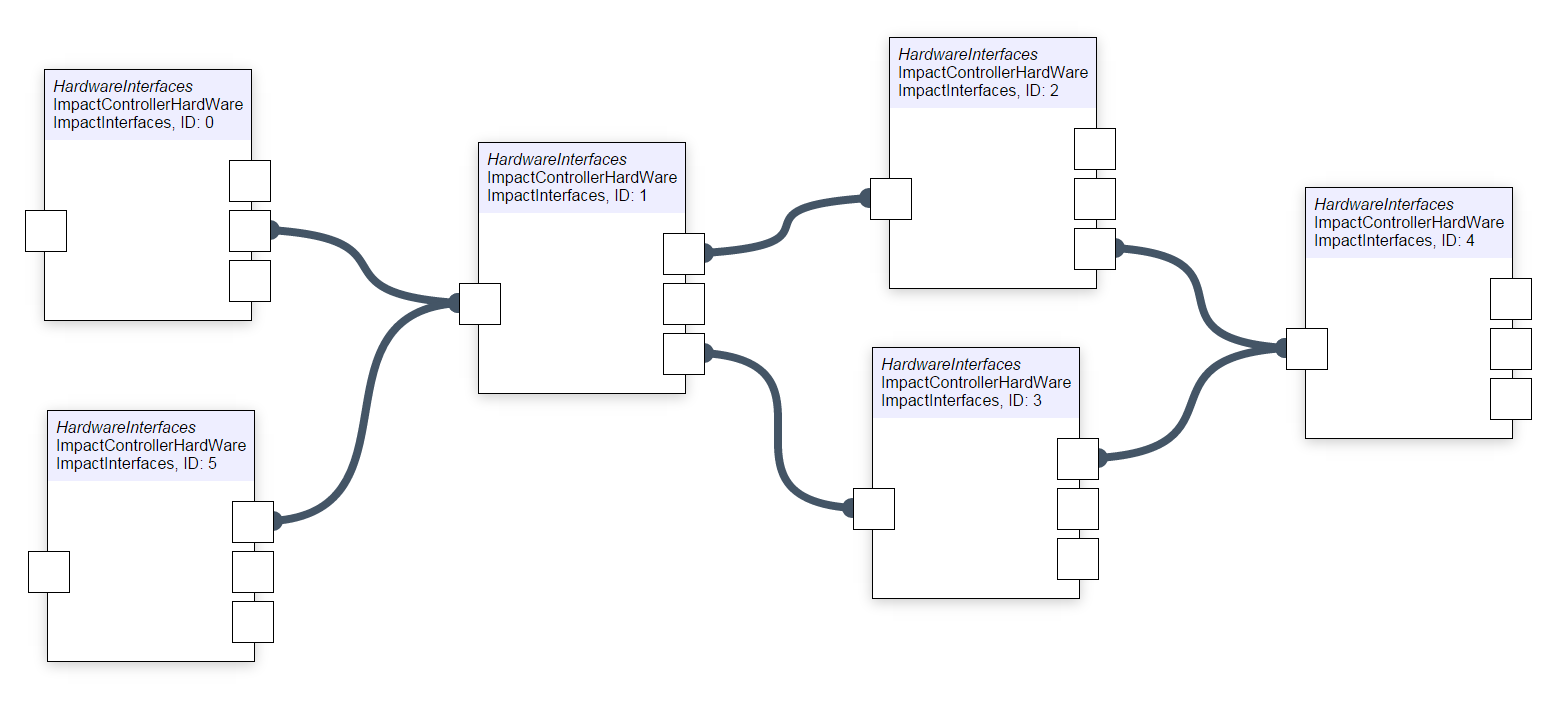


Figure 3: Prototype of the IMPaCT Toolkit build with jsPlumb

jsPlumb is an open source (dual licensed as MIT and GPLv2) JavaScript library for visually connecting components in web applications (see Figure 3). This can be used as a tool to develop a visual graph editor. jsPlumb seems to be well documented.

However, unlike JointJS, jsPlumb is only a visualization tool and does not implement a Model-View-Controller framework. Building these ourselves would require more effort. Because of this, the decision was made in favor of JointJS.

#### NoFlo

The MIT-licensed NoFlo library can be used as a library for making complex workflows more manageable (see Figure 4). It has been written in CoffeeScript for simplicity. The user interface of this library is very modern and it can be used on desktops and tablets. NoFlo developed its own protocol for communicating with runtimes (such as sever-to-server or server-to-microcontroller), namely the FBP Network Protocol.

Currently there is very limited documentation of the NoFlo library. This requires going through the source code and limited number of examples to find out how to implement this for the IMPaCT Toolkit. When adding a large number of nodes (around 50 or more), there were some performance issues in some web browsers. Based on this, we decided to not use the NoFlo library.

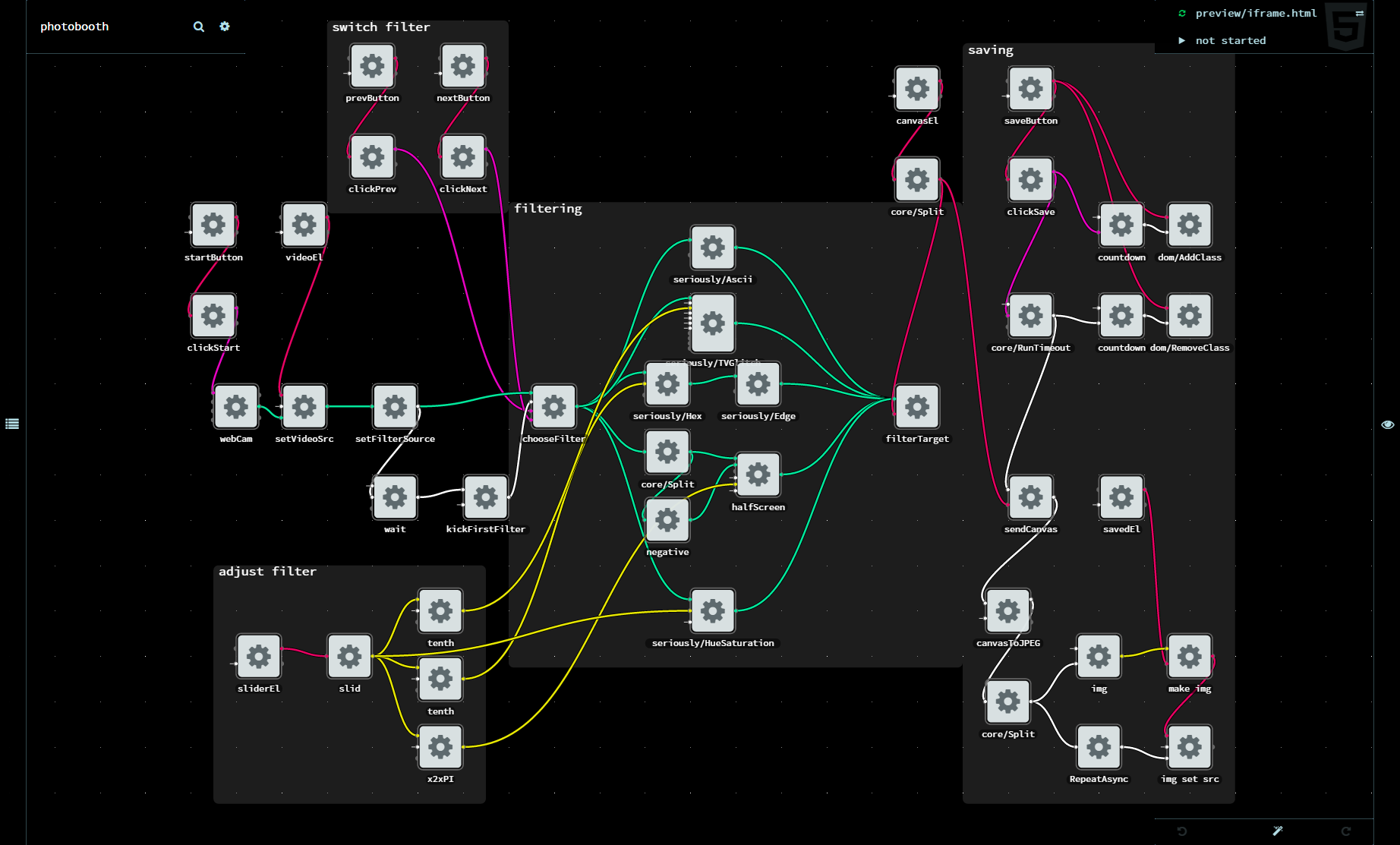


Figure 4: Prototype of the IMPaCT Toolkit build with NoFlo

#### Eclipse GEF (Graphical Editing Framework)

Eclipse GEF (Graphical Editing Framework) is a framework for developing graphical editors on the Eclipse Rich Client Platform (Eclipse RCP), written in Java (see Figure 5). This platform facilitates the development of editors as a stand-alone application.

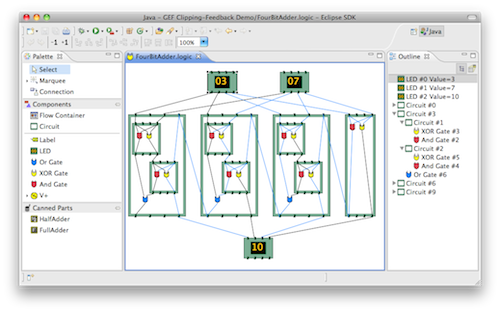


Figure 5: Example of an Eclipse GEF application

Eclipse GEF seems to be well designed and well documented. However, we gave the preference to use web technology to develop the IMPaCT Toolkit, so it would become easier to port the application to a web application in the future and to ease the addition of configuration assistant systems.

### Decision: Usage of JointJS, Zurb Foundation, jQuery, Jackson, and Jetty

For a fast implementation process of the IMPaCT Toolkit several third-party software components have been used. These have been chosen according to their practical usability within this project.

* **JointJS** is used for the JavaScript implementation parts, such as the frontend. JointJS uses **Backbone** and **Underscore**.
* **Zurb Foundation** is used for the user interface parts, i.e., the frontend. It is comparable to Twitter Bootstrap, another user interface library.
* **jQuery** is used for JavaScript implementation parts.
* **Jackson, Jersey** and **Jetty** are used for the REST server and backend implementations.

# Toolkit Prototype

## Overview



Figure 6: Deployment diagram of the IMPaCT Toolkit

We have decided to develop the IMPaCT Toolkit as a stand-alone application. We did however make use of Web technology, since there were good libraries available (JointJS) and because this makes it easy to port the toolkit to a web application in the future.

The tool runs as a small webserver (facilitated by Jetty) on the machine of the end user. The back-and, where the model (the configuration) is stored and validated, runs on top of this small webserver. The model is built using the Eclipse Modelling Framework.

The front-end (user interface), which can be used to modify the model (thus the configuration for the IMPaCT Controller) runs inside a web browser on the same machine. The front-end and the back-end communicate using a simple, custom REST API. The final configuration can be loaded from the IMPaCT Toolkit to the IMPaCT Controller. A deployment diagram of the IMPaCT Toolkit can be found in Figure 6.

## Configuration Data Model

### Aspects of the Configuration Model

The following aspects have to be covered by the configuration model:

1. Technical port configuration: Configuration of hardware interfaces of the Controller (“wiring”)
2. External port configuration: Configuration of hardware and software interfaces of the controller to external entities (e.g., DSM, VPP, …)
3. Additional (meta-) information about the devices connected to the hardware interfaces (e.g., technical specifications of CHP); This information is necessary to enable a successful control of the devices by the external entities

### Configuration Model Architecture

To model the configuration architecture an EMF model is used. Eclipse EMF can be used to model a domain model. EMF has a distinction between the meta-model and the actual model. The meta-model describes the structure of the model. A model, which e.g. contains the actual configuration of a distinct device, is then an instance of this meta-model.

The basic idea is to use a data driven architecture to represent the device consisting of components with inputs and outputs which are connected by wires. The meta-model of this architecture is shown in Figure 7.

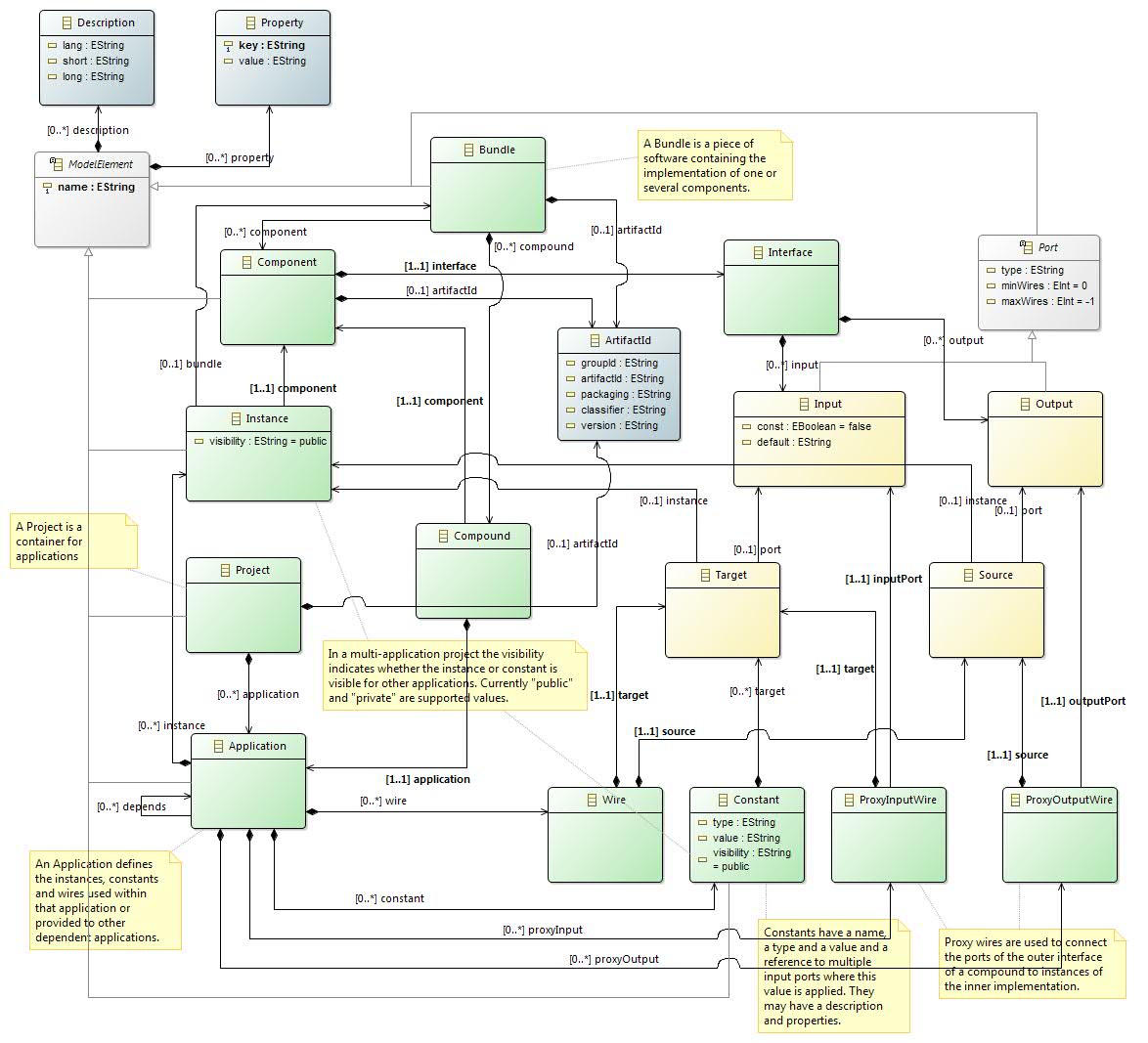


Figure 7: EMF meta-model for the IMPaCT Toolkit

The concepts of this meta-model and the classes are described in the following chapters.

#### Model Entry Points

There are three main entry points into the model:

* A **Component** is the metadata description of an instancable function block containing mainly its interface with input and output ports. It is used in the editor where components and especially their instances are visualized including their interface, but may also be used at runtime or for code generation.
* A **Project** is the result of the construction process performed in the editor. It acts as a container for Applications, which then contain instances of components, constants and the wiring between them.
* A **Bundle** is a piece of executable software implementing one or several components. It is deployed on a runtime by some means which are outside the scope of this model.

Each entry point can be used to create an instance of the model, called an artifact, which can be stored in a separate file.

#### ModelElement

A ModelElement is used as a generic superclass for objects which can be referenced by some other object. The ModelElement therefore contains a name which is used for the reference. It also may contain a description for the human user and a set of arbitrary properties which can be used for various purposes.

#### Description

The description contains information for the human user. There may be a short and a long description. Multiple descriptions can be attached to a model element, e.g. in multiple languages.

#### Property

A Property is a simple key/value pair storing auxiliary information on a model element. The meaning of the properties is not defined by the model.

#### ArtifactId

An ArtifactId is used to uniquely identify instances of the three main entry points of the model. It can be used to locate these instances. For example, there may be a component repository where existing component definitions are stored and can be retrieved. Similarly, there may be a repository for the implementing bundles or for the configuration projects.

#### Component

The first step for the creation of a configuration is the definition of the components used. A Component mainly contains an interface describing its external appearance.

#### Interface

The Interface of a component consists of its input and output ports.

#### Port

A Port is the generalization of inputs and outputs. Ports have a name and a type as well as constraints about the number of connected wires. The type can be used to verify that only matching ports are connected.

#### Input

An Input is a specialization of Port which consumes values. It can have a default value which is applied when no wire is connected. An Input also can have the property const=true. In this case, only a Constant may be connected to this input port and not an arbitrary wire. Constant inputs resemble configuration parameters for a component which cannot be changed during runtime.

#### Output

An Output is a specialization of Port, which generates values.

#### Project

A Project is the second main entry point into the model. While the Component and its related classes represent the abstract definition of possible building blocks, the project represents the actual configuration of a specific device. A project mainly acts as a container for Applications.

#### Application

Applications are used to structure a project into several parts, which can be combined to form an overall project. They are a very important part of the model, as they contain

* the instances of components used in the application
* the wires used to connect the ports of the instances
* the constants used to represent configuration parameters
* The proxy wires used in compound components (see below for further explanation)

#### Instance

An Instance is the entity representing the instance of a component. An instance may optionally also refer to a bundle containing the implementation of that instance.

#### Wire

A Wire connects a Source representing an output port producing values to a Target representing an input port consuming values.

#### Source

A Source is a reference to an output port of a specific instance.

#### Target

A Target is a reference to an input port of a specific instance.

#### Constant

A Constant represents a constant configuration value, which can be applied to one or multiple targets.

#### Bundle

A Bundle is the third entry point into the model. It represents a bundle of executable software, which implements one or more components. Bundles may refer e.g. to a Java jar archive or a dynamic library (dll, so) or other. Bundles may however also implement components by means of so-called compounds.

#### Compound

A Compound is an implementation of a component, which uses nested wired components. The implementation is defined in an application within the compound. To connect the ports of the outer interface of a compound to ports of the inner instances of its implementation, so called proxy wires are used (see below).

#### ProxyInputWire

A ProxyInputWire links an input port of the outer interface of the component to an input port of an inner instance of the compound implementation of that component.

#### ProxyOutputWire

A ProxyOutputWire links an output port of an inner instance of the component to an output port of the outer interface of the compound implementation of that component.

## Architecture

In Figure 8 an overview of the IMPaCT Toolkit is displayed.

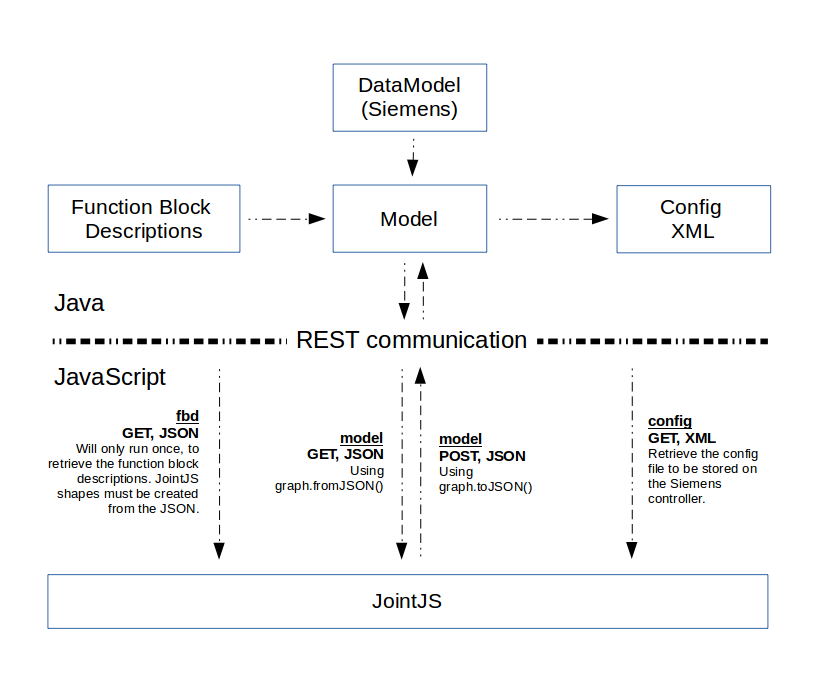


Figure 8: Architectural overview

## Software Components

### Third-party Components

The implementation of the IMPaCT Toolkit uses several third party software components:

JointJS is used for the JavaScript implementation parts.

* License: Mozilla Public License Version 2.0

Zurb Foundation is used for the frontend.

* License: MIT license

jQuery is used for JavaScript implementation parts.

* License: MIT license

Jackson, Jersey and Jetty are used for the REST server and backend implementations.

* Jackson licenses: ASL, LGPL
* Jersey license: CDDL 1.1
* Jetty license: Eclipse Public License 1.0

### Developed Components

For the frontend of the IMPaCT Toolkit, we can distinguish three components that were developed by ourselves: JointJS shapes, the palette and the inspector.

#### JointJS shapes

JointJS allows developers to create new shapes by extending the basic shape. For the IMPaCT Toolkit we created shapes for components (see 2.2.2.6), constants (see 2.2.2.17) and compounds (see 2.2.2.19). In Figure 9 the graphical representation is displayed.

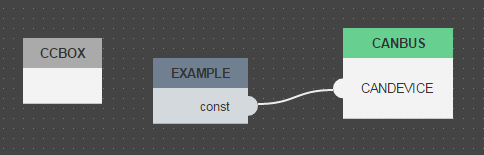


Figure 9: The three custom shapes that were developed

#### Palette

The palette (see Figure 10) allows users to create instances of the JointJS shapes mentioned in the previous paragraph. Through an AJAX call, the components are fetched from the Java backend.

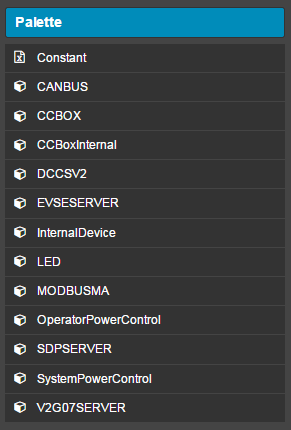


Figure 10: The palette for the IMPaCT Toolkit frontend

When the user clicks on an item in the palette, a dialog is shown in which parameters can be specified. Figure 11 shows an example of such a dialog, in this case for components. The user has to specify a name and can add the component as compound if desired.

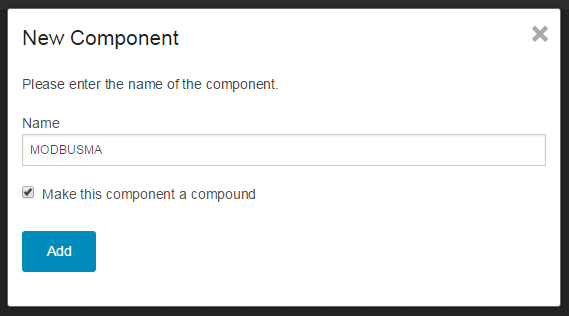


Figure 11: The dialog that is shown when adding a component

#### Inspector

The inspector allows users to change the properties of the project (see 2.2.2.11), applications (see 2.2.2.12) and constants (see 2.2.2.17). The user can add properties, remove properties and edit the values of them (see Figure 12).

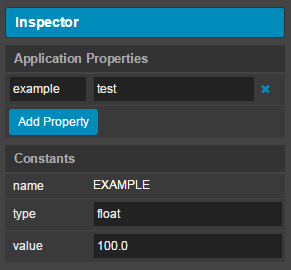


Figure 12: The inspector for the IMPaCT Toolkit frontend

# Evaluation

In the evaluation phase of the IMPaCT Toolkit, the implementation has been checked with regard to the requirements of the two scenarios:

* Charging Station Scenario
* Smart Building Scenario

The evaluation of the Charging Station Scenario took place on the 9th and 10th of December 2014 at Siemens in Munich. Since the focus of this year was on the Charging Station Scenario, no hardware evaluations of the toolkit with respect to the Smart Building Scenario could be performed. Instead a qualitative evaluation was performed.

## Evaluation: Charging Station

During the meeting in Munich on the 9th and 10th of December 2014, several tests did take place. Most importantly, the testing of the integrated system with backend, frontend, and IMPaCT Controller has been conducted.

The evaluation was successful. All components integrated well into the system. The configuration file was successfully manipulated in the IMPaCT Toolkit and sent to the IMPaCT Controller that was connected to a charging station see (Figure 13).

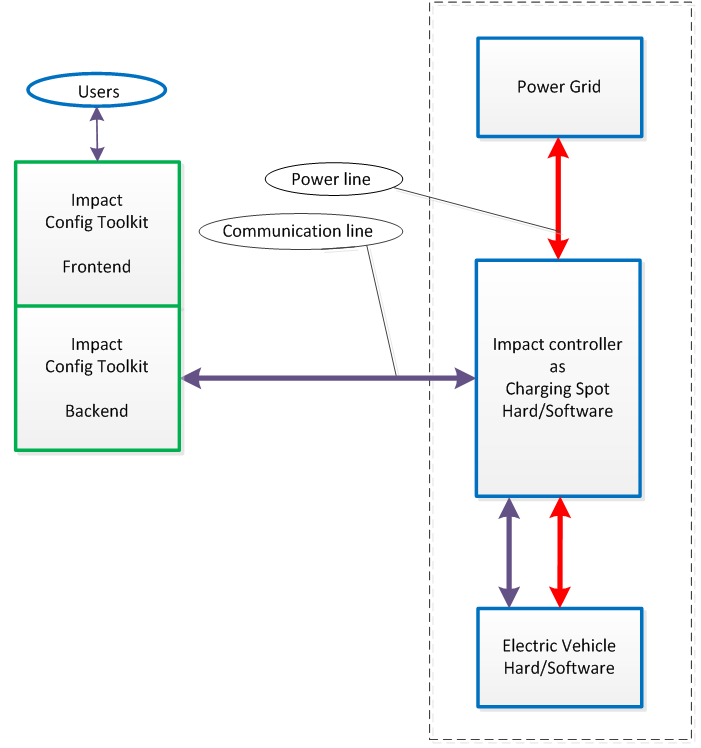


Figure 13: Tested scenario with the Charing Station

### Configuration of IMPaCT charging spot controller

The IMPaCT charging spot controller is based on an existing product for different kinds of charging equipment. It uses its own proprietary configuration files. To adapt to this controller, an adapter was developed, which translates from the common IMPaCT configuration model to the proprietary file format.

The following two paragraphs describe the project that was created in the IMPaCT Toolkit frontend, and the processing that is done in the backend.

#### Frontend project

For the charging station test, we created a new project (see 2.2.2.11) in the IMPaCT toolkit frontend. The project has one root application (see 2.2.2.12) shown in Figure 14. It contains five constants (see 2.2.2.17) and two compounds (see 2.2.2.19). The MAXVOLTAGELIMITSYSTEM is the most interesting constant, which is further described in paragraph 3.1.2.

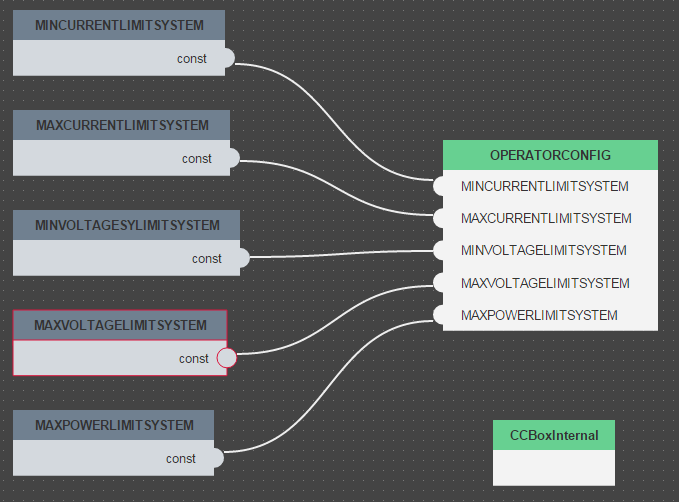


Figure 14: The root application for the charging station project

The OPERATORCONFIG compound contains nine constants and one component, as shown in Figure 15.

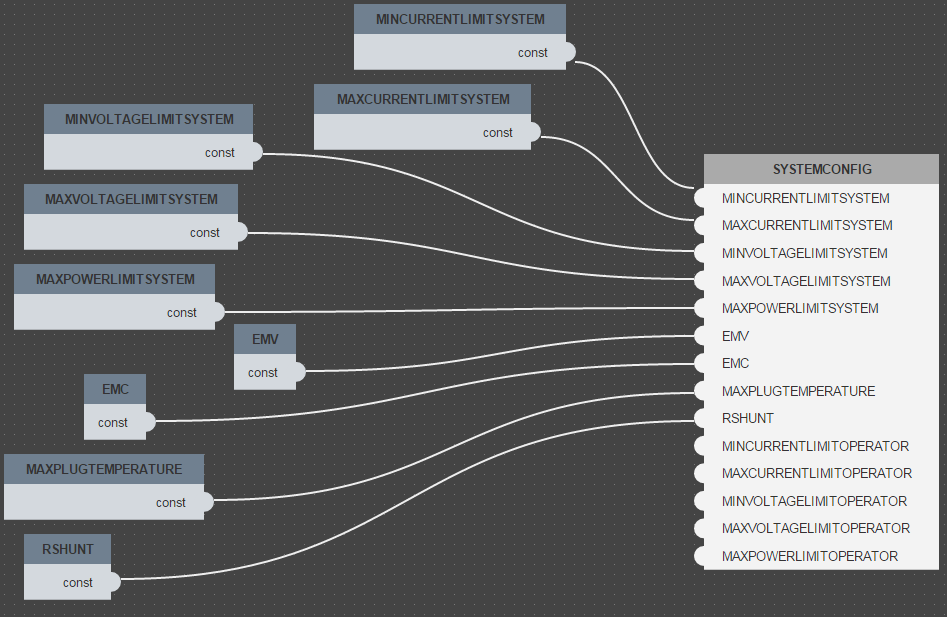


Figure 15: The implementation of the OPERATORCONFIG compound

Finally, the CCBoxInternal compound contains 28 constants and 10 components. Figure 16 shows the effectiveness of constants. Some of the constants are reused for more than one port (see 2.2.2.8), for example the INVALIDTIME constant. The BLINKTIME constant will also be used for the test, which is further described in paragraph 3.1.2.

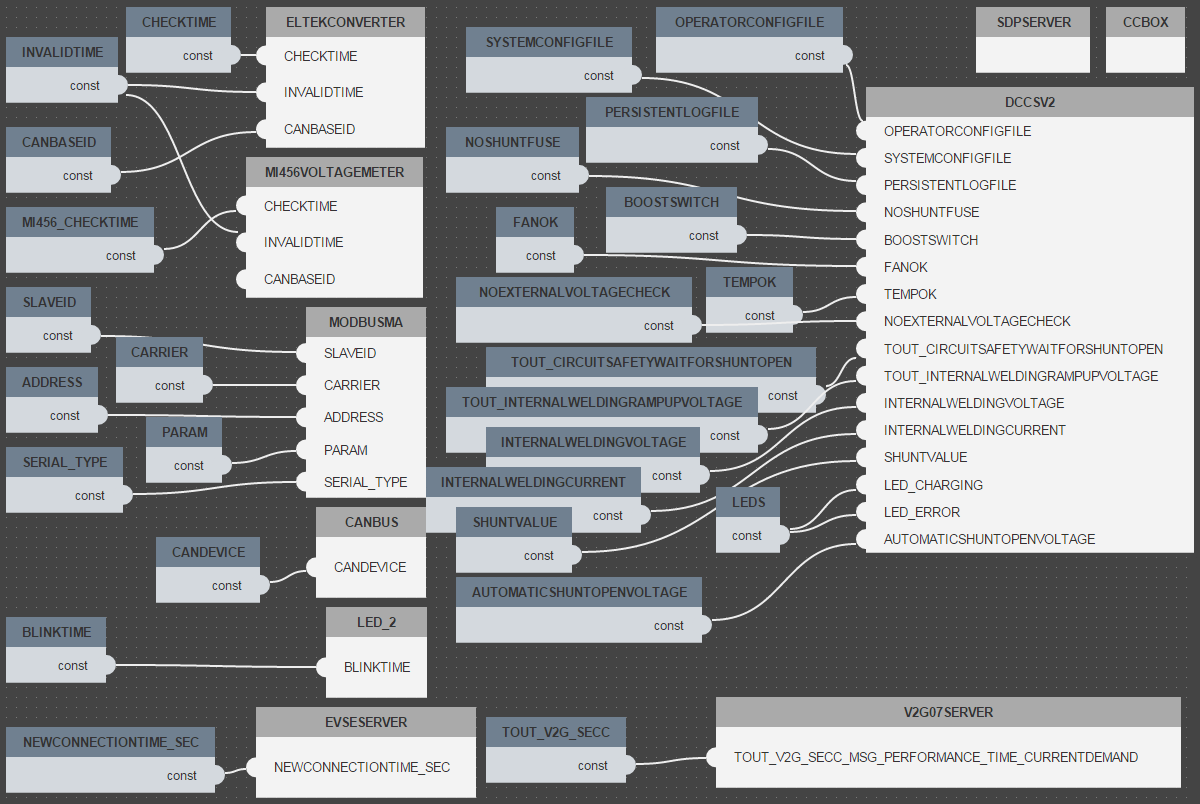


Figure 16: The implementation of the CCBoxInternal compound

#### Backend processing

When the configuration model is updated via the graphical user interface, the adapter automatically provides a notification to the controller, which then fetches and applies the new configuration files.

The following is a snippet of the log file during the deployment of a new configuration:

2014-12-10 11:26:53.933:INFO: Model changed

2014-12-10 11:26:53.933:INFO: Deployment of new configuration triggered at 20141210112653

2014-12-10 11:26:53.933:INFO: Save project test

2014-12-10 11:26:54.288:INFO: Model updated, returning list of config files

2014-12-10 11:26:54.478:INFO: Downloading config file dccsv2.cfg

2014-12-10 11:26:54.682:INFO: Downloading config file dccsv2\_operator.cfg

2014-12-10 11:26:54.933:INFO: Downloading config file dccsv2\_system.cfg

2014-12-10 11:26:57.271:INFO: Update successful for timestamp 20141210112653

2014-12-10 11:26:57.272:INFO: Latest update installed

The log file starts with the event that the configuration model has changed. It then triggers the deployment of the new configuration and at the same time stores the changed model on disk.

A few milliseconds later, the IMPaCT controller is informed about the update and gets a list of affected configuration files. It then downloads them step by step.

Finally, the IMPaCT controller installs these configuration files and sends an acknowledgement to the backend. After that, the latest update is installed.

### Parameters configured during evaluation

During evaluation, only some exemplary parameters could be configured. We have selected the following two:

* The maximum voltage of the charger
* The blink frequency of an indication LED

The test itself was performed using a test device which behaves as a car and which could be used to perform a charging process. We measured the power consumption of the charger to monitor the effect of limiting the charging voltage. In addition, the charging voltage was displayed at the test equipment.

### Test execution

The test was successfully executed on Wednesday the 10th of December at Siemens in Munich. For testing purposes we ran the frontend (JavaScript) and backend (Java) on two different computers, as can be seen in the figure below. Both computers and the charging station were connected to the same switch (see Figure 17).



Figure 17: On the left the frontend, on the right the backend

The initial value for a charging session was set to 250 Volts. A display allows us to verify the value that is being used, as seen in Figure 18



Figure 18: The initial voltage was set to 250 Volts

After changing the MAXVOLTAGELIMITSYSTEM parameter to 200 Volts in the frontend, the value was updated in the backend and sent to the charging controller. Figure 19 reflects this.

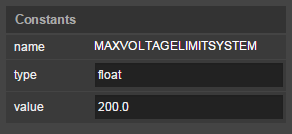
 ****

Figure 19: The updated voltage after changing it in the IMPaCT Toolkit frontend

The blink frequency of the indication LED was also tested with values ranging from 100 milliseconds to 2000 milliseconds. This test also succeeded.

## Evaluation: Smart Building

Since the focus of this project year was on creating a product in the charging station scenario, no hardware test was possible in the smart building scenario. Nevertheless, the ability of the IMPaCT Toolkit to configure smart buildings was tested. This was done by using the IMPaCT Toolkit to configure a reference design based on the Energy Smart Home Lab on the Campus of the KIT (see Figure 20 and Deliverable AD0038 Section 2.2.4). In addition, the FZI House of Living Labs (see Figure 21) was used as a test bed.



Figure 20: KIT Energy Smart Home Lab

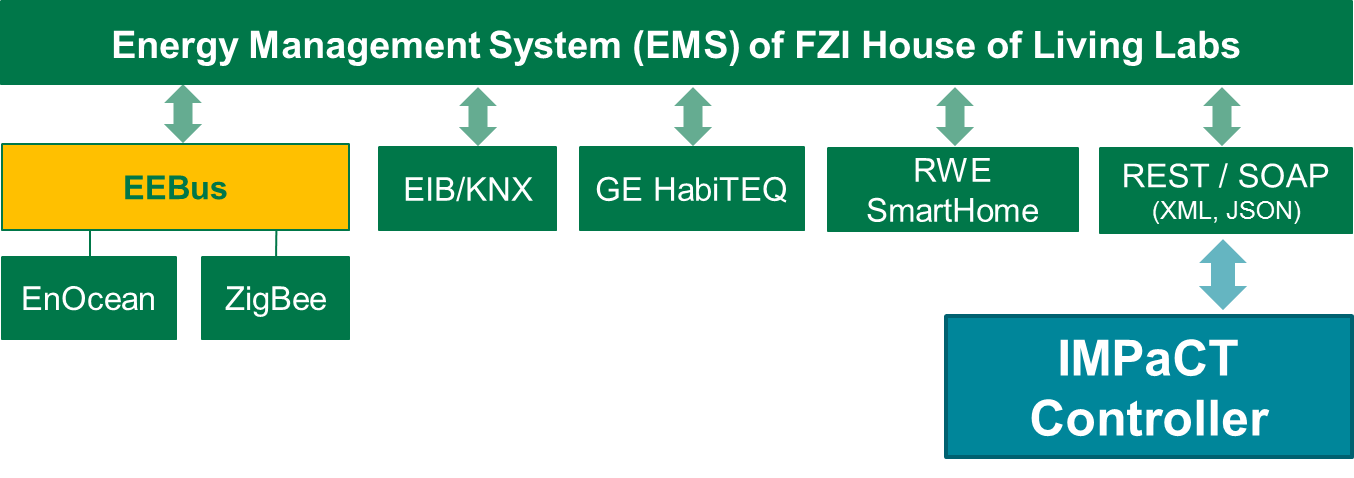


Figure 21: Setup of FZI House of Living Labs

The configuration files that have been built in both cases contain the configuration of the IMPaCT Controller, which is necessary to work properly in the respective environment. Figure 22 and Figure 23 show the user interface of the IMPaCT Toolkit during the configuration of the KIT test bed.

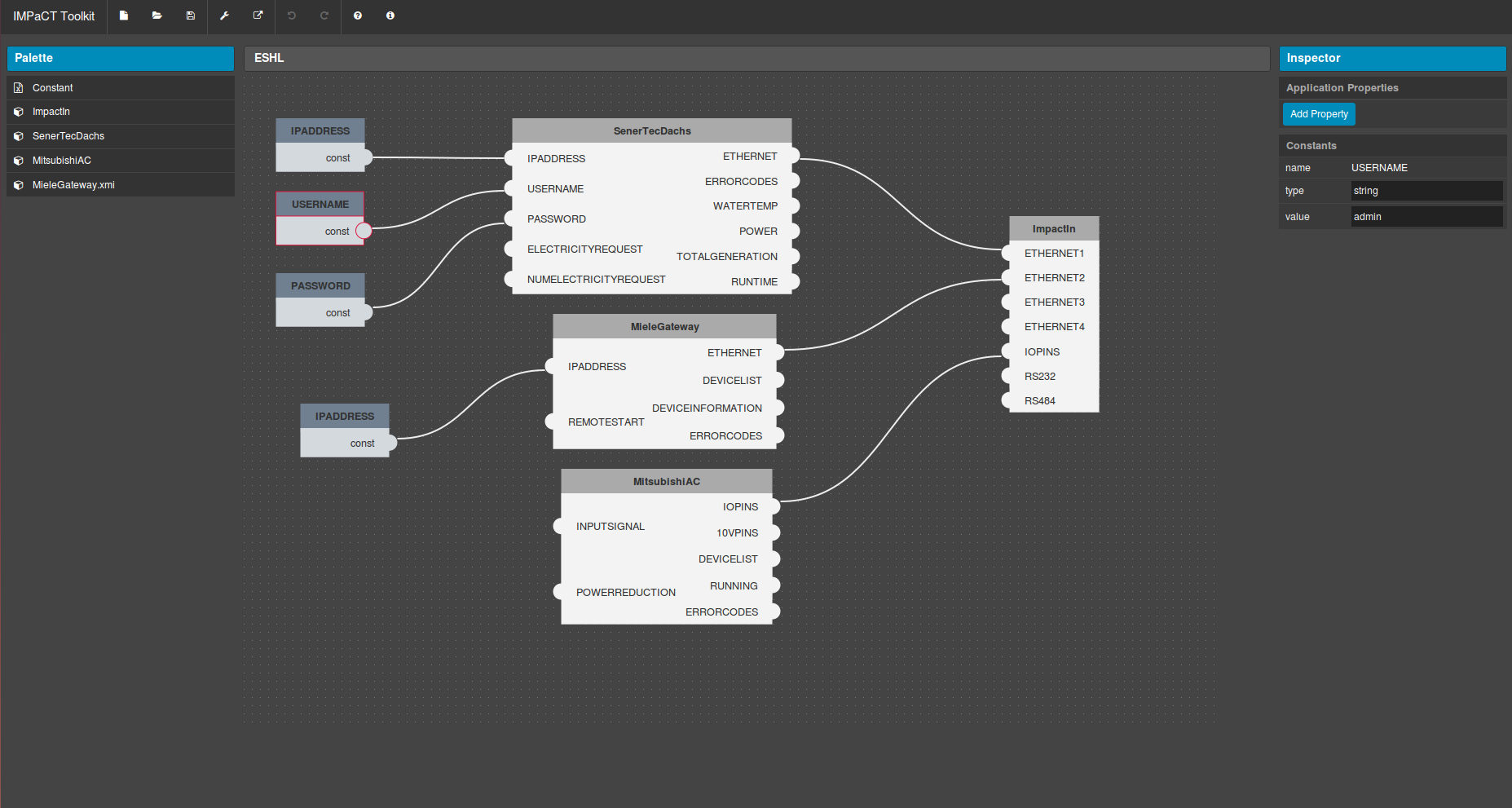


Figure 22: IMPaCT Toolkit configuration of “KIT Energy Smart Home Lab” reference design

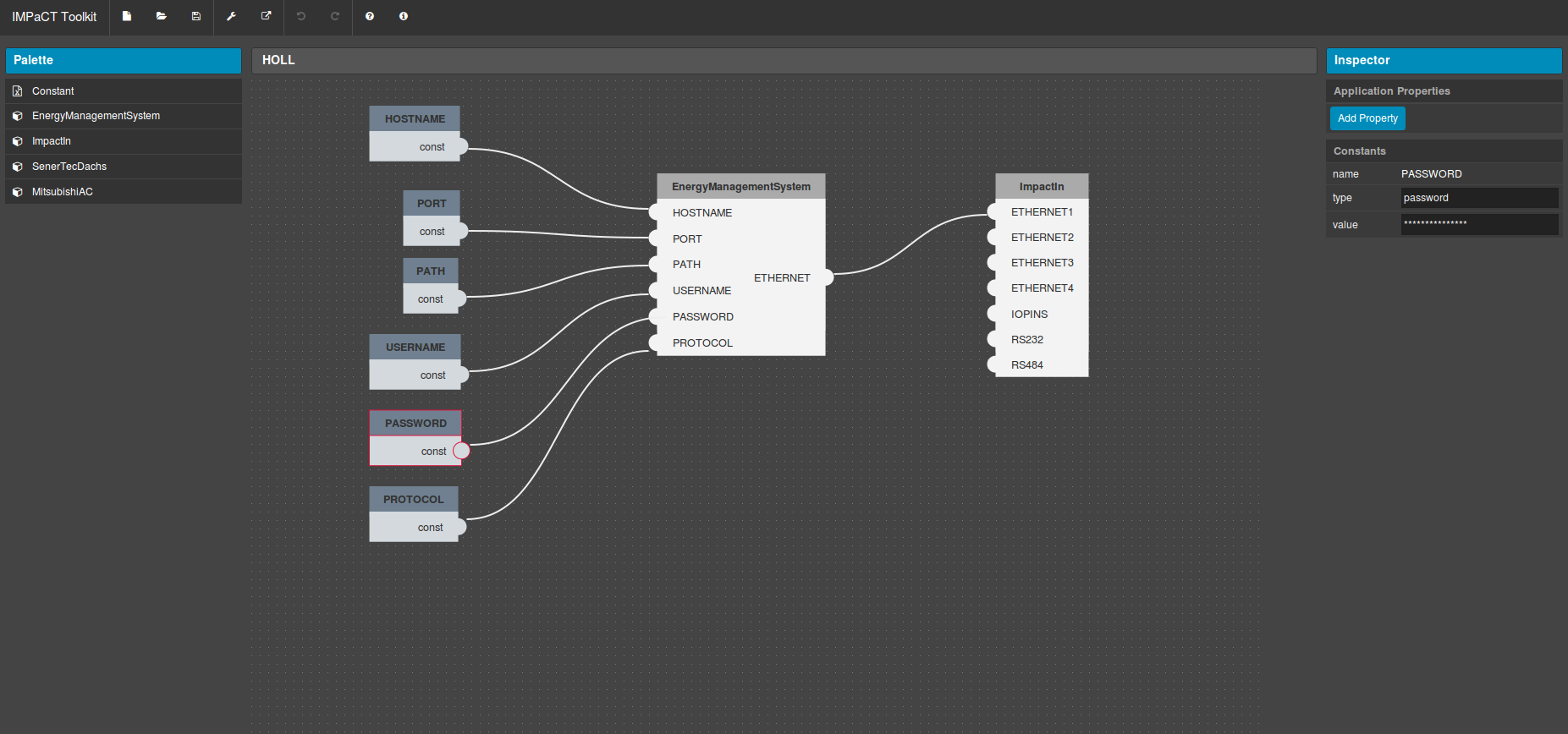


Figure 23: IMPaCT Toolkit: Configuration of the Smart Building “FZI House of Living Labs”

These configuration files can now be saved to a controller that can now easily be installed into Smart Buildings and configured properly with respect to the local devices and requirements for communication with higher systems, such as the participation in a Virtual Power Plant.

The test described above shows that in principle the IMPaCT Toolkit could be used to configure the IMPaCT controller in a smart building scenario. The interaction with the IMPaCT Controller, the IMPaCT Toolkit and the respective smart building has to be tested. To be able to use the Toolkit more efficiently with respect to other communication hardware interfaces used in Smart Buildings, the addition of a more sophisticated user interface is needed. This will be addressed in 2015.

# Conclusions and Outlook

The IMPaCT Toolkit worked well in the evaluated scenarios: Charging Station and Smart Buildings. Configuration files can now easily be created by clients using the IMPaCT Toolkit for the configuration process of the IMPaCT Controller.

This demonstrated that the developed system is now able to enable a practical way for configuring complex systems, such as the controllers in the energy domain. The GUI can be customized, additional pre-defined configurations, i.e., compounds, are automatically when being loaded into to IMPaCT Toolkit, and complex configurations can be stored for later re-use in other systems, which may be even more complex.

All in all, the IMPaCT Toolkit is an important and valuable contribution to the controller of Siemens that enables all users to configure their controller with respect to their requirements.

The IMPaCT Toolkit successfully showed its applicability and different scenarios. In 2015, the IMPaCT Toolkit will be extended to further scenarios, such as Micro Smart Grids, and, more importantly, by user assistance functionality. This additional functionality will work as a wizard that leads the users through the full configuration process of the IMPaCT Controller. It will contain best practice recommendations, helping tooltips, and selection mechanisms of pre-defined compounds and configurations.