End-User Service Compositions in the Internet of Things

Author: Jonas Hartweg

Supervisors: Dr. Pedro Valderas (UPV), Dr. Victoria Torres Bosch (UPV)

Examiners: Prof. Dr. Klaus Jung (HTW), Dr. Pedro Valderas (UPV)

Universities: Universitat Politècnica de València, HTW Berlin

# Abstract

…

# Table of contents

Abstract 2

Table of contents 3

Introduction 5

The State of the Web 5

The Internet Of Things 5

End-User Service Compositions 6

Goal and Motivation 7

Ethical Considerations 8

Creating a sense of moral responsibility among developers, service providers and users 8

The power of big data 8

Usage and distribution of open-source software 9

Background 10

The Internet of Things 10

The Web of Things 11

Business Process Modeling 11

End-User Development 11

Change Patterns 11

State of the Art 12

Research Context 13

SITAC 13

Service Layer Module 14

Services Invocation 15

Services Registry 15

Services Composition 15

Analysis and Design 16

Domain Specific Language 16

Abstract Syntax 16

Concrete Syntax 18

Implementation 21

The Frontend 21

Frontend logic with JavaScript 21

Layout and Design 24

Building a web app 26

Development workflow 27

Software stack and debugging 28

The Backend and BPMN generation 29

Data structure and conversion 30

Conclusions 32

Limitations and Future Work 32

Literaturverzeichnis 33

Table of Figures 35

Appendix 36

**Terminology**

IoT: Internet of Things

Web application: Interchangeably used with web page or website.

SITAC: Social Internet of Things - the research context of this work.

Activity: An instance of an object in the Internet of Things

Service: Interface to interact with objects in the Internet of Things

Service composition: A set of configured activities and fragments in a process flow

Fragment: Conditional statements that helps to compose Activities based on certain preconditions

(Conditional) Branch: A service composition that is executed based on a certain precondition

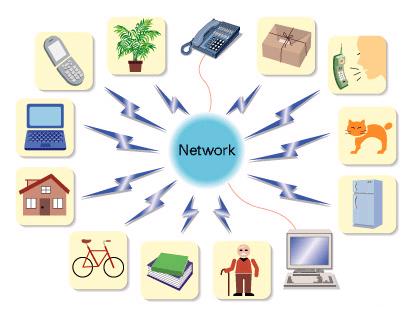
# Introduction

## The State of the Web

The web is transforming. Once a medium to serve static web­pages, delivering the same information to any user via a web browser a process coined “Web 2.0” and the availability of smartphones and tablet computers changed the way we conceive the Internet.  User collaboration and the ever-present access to the knowledge of world through mobile devices led to a ubiquitous, unprecedented connection of information that became an integral part of the society we live in. While internet services used to limit access to their data only to their own web pages they soon realized that open information and granting access to third parties will spread knowledge, make room for innovation and makes possibility of combining multiple services into new ones. Facebook for instance, which offers an open software platform building on their social network, makes it possible for game companies like Zynga to develop social games and also Google, which with its personal assistant Google Now taps into sources like Wikipedia, newspapers or sports websites to deliver relevant, aggregated information to the user.

While the cyberspace is already a heavily linked and interconnected place, the next step in the evolution of the Internet will be the interconnection of real-world objects.

## The Internet Of Things



[[1]](#footnote-1) Whether its appliances, buildings or even people ­ everything can become a part of the Internet to create the so-called “Internet of Things” (IoT). With the help of smart sensors and radio transmitters real-world objects can become smart objects that can be interacted with using Internet services. This can be the connected fridge, which tells a person whether the milk is going bad or the student who checks the universities library web service whether their are free seats available. Also medical applications are possible. Embedded heart rate monitors for cardiology patients or blood glucose meters for diabetes patients are already changing the way we think about health care.

Key areas that will be affected by the Internet of Things are health, logistics, home automation and transportation. However, the possibilities are limitless, since any physical object can easily transformed into a device on the Internet of Things. It merely needs a sensor, a processor and some means of communication to participate in the IoT.

This hardware components shrink in size and cost as …

(Statistics)

While these are future prognoses, the Internet of Things already arrived in enterprise environments. …

It is time to build tools to manage this development and give end-users a way to utilize the IoT’s benefits.

## End-User Service Compositions

Service compositions are a combination of multiple services into a single one, creating a richer experience and higher functionality. It helps end-users to automate reoccurring tasks that they might perform every day. An example of this could be automating home control. A user could set up that every time he comes home the door unlocks, lights turn on and radiators heat up.

The following example describes an everyday situation of a university student that can be modeled in form of a service composition. It is used throughout the paper and is based on concept paper done at the Universitat Politècnica de València (Valderas, et al. 2013).

John is an architecture student at the Universitat Politècnica de València (UPV). During the examination period he spends his days in one of the university’s libraries where he prepares for upcoming exams. He is not the only one preparing for tests, which is why the libraries are usually full and it is hard to get find a seat.

His usual routine is to take bike from on of Valencia’s bike sharing stations to go the university campus. After arriving he looks for bike station that has free spots, so he can hand in the bike. Next up he walks from library to library to find a seat. When he found one with empty seats he asks his fellow students via Facebook to join him.

This is a time consuming and tedious task he performs every morning. Using services of the Internet of Things, it can be automated.

To do this, the process has to be split up into multiple steps:

1. Find a library with empty seats
2. Search for the nearest bike station with free spots
3. Book a spot at the bike park and a seat at the library
4. Optionally, inform friends via Facebook

It requires access to a library service, which lets a student search and book empty seats in available libraries, a bike service, which lets the student check and book a spot at free bike stations and finally access to Facebook to message his friends.

For a formal description of this use-case the Business Process Model Notation, or BPMN, can be used. This is a notation to illustrate business processes, but can also be used to describe this everyday situation.

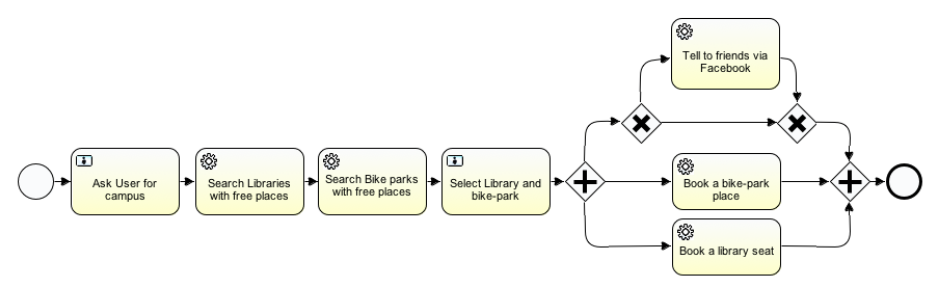


Figure 1: Use-Case BPMN (Source: End-User Service Compositions in the IoT)

(More on BPMN and Change patterns)

## Goal and Motivation

The first challenge of this work to build a mobile web application to consume and combine services offered by connected objects, so called “smart objects”.

Goal is to develop a smart client[[2]](#footnote-2) with a usable and consistent interface and to build a state-of-the-art mobile application with the help of web technologies like HTML5, CSS3 and JavaScript that enables the user to create these service compositions. It is required to perform as well as a native application and therefore utilizes new APIs that make it possible to interact the native software and hardware of mobile devices to create a native-like experience using web technologies.

The tool is targeted at users without a programming background, which is why this work touches the research field of End-user Development. To give a focus to the vast field of the Internet of Things, end-user targeted topics like smart buildings and home automation will play the central role. Change patterns are utilized to simplify the workflow of linking activities.

The application is required to work in conjunction with a web server that delivers service information.

The second challenge is to export these user-generated service compositions in form of an executable business process model. To actually get use out of the modeled service composition, it has to be deployed to a so-called BPM system. This BPM system automatically executes all steps of a service composition by communicating with the activities’ services and, when needed, interacts with the user for further input.

For service execution, the BPM system demands a Business Process Model as input. This will be generated on a server, based on the service composition that the front-end delivers. Since the final system of executing activities isn’t ready for production yet, this step will mostly serve as a simulation. The system will test, whether modeled service compositions are valid BPMs and whether they are suitable for later deployment in the running SITAC system.

This project builds on prior work done by SITAC[[3]](#footnote-3), the Social Internet of Things, in the context of ITEA 3[[4]](#footnote-4). SITAC is an ecosystem of network-connected entities, which provides a platform to interact with smart objects in the Internet of Things. ITEA 3 is a program to stimulate technical research and innovation in Europe. Corporations, research institutes as well as universities support its projects.

## Ethical Considerations

The following will explain moral considerations that shareholders have to address when dealing with sensible user data. Furthermore, the usage and distribution of open-source software is being outlined.

### Creating a sense of moral responsibility among developers, service providers and users

Shareholders of data-driven software have to be aware of the potential power about the ability “to combine data from various seemingly insignificant activities to create potentially significant profiles” (Covert und Orebaugh 2014). This begins in schools, where media competence is taught and ends in companies that offer workshops to their employees to teach media literacy and the implications of handling sensitive user data.

To ensure certain moral guidelines to provoke ethical behavior of their employees companies might opt to define a code of conduct[[5]](#footnote-5). This code can define how employees should treat clients, users and co-workers and how they can use their work to have a positive impact on society.

### The power of big data

Developers, providers and users of data-driven software and services need to be aware of the responsibility that comes with the collection of big amounts of user information. The power and potential for misuse of “big data”, such as user information gathered through IoT devices, requires a sense of moral responsibility from all parties that have access to this data. Shareholders need to be educated about the dangers that lie in handling sensible user data. Whether these are threads from governments, insurance companies, marketing firms or private people that have interest in access to sensible information.

For instance, revelations by Edward Snowden about the US-American National Security Agency’s “Prism” program have shown the demand by government agencies of insight into its citizen’s lives and the extreme measures taken to achieve this. Companies like Microsoft, Apple and Dropbox are targets of warrant-less requests by the NSA to give access to its user’s data. According to the Guardian “the NSA claimed to have "direct access" through the Prism program to the systems of many major Internet companies, including Microsoft, Skype, Apple, Google, Facebook and Yahoo.” (Greenwald, et al. 2013) It reports that the companies’ communications are allowed to being “collected without an individual warrant if the NSA operative has a 51% belief that the target is not a US citizen and is not on US soil at the time” (Greenwald, et al. 2013).

Developers need to put systems in place that protect user data from malicious threads, like hackers, by using encryption and by only collecting the absolute minimum amount of data that is required to run the service they are providing. Being transparent about how user data is used builds confidence about using data-driven services.

Individuals that are working for companies that deal with user data – especially, if this can be sensible information like health data, collected by IoT devices – have to be aware of these threads to their user’s privacy and have the duty to protect the user’s identity. Eventually, the users own their data and therefore, this data can only be processed in the confines of mutually accepted terms and conditions. Companies have the duty to inform their users, when personal data is being shared with third parties. Whether these are governments that act under the pretext of national security or corporations that are driven by financial interests. Ed Covert and Angela Orebaugh from SC Magazine note that “just because something is good for the aggregate does not necessarily mean it is good for the individual” (Covert und Orebaugh 2014). Finally, they propose a utilitarian approach when dealing with the Internet of Things to maximize the good that results out of its applications.

### Usage and distribution of open-source software

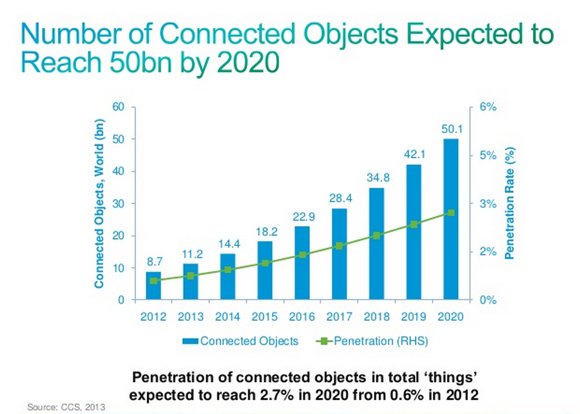
This work makes heavy use of open-source software. Among the list of open-source software used are Grunt.js, Node.js, all front-end libraries and the source-code editor Atom. When using community products it advised to contribute as well by helping to improve open-source software or by disclosing the source-code of own software.

# Background

## The Internet of Things

The term “Internet of Things” describes the expansion of the World Wide Web into the physical world. Advances in the miniaturization of chips and integrated circuits make it feasible to integrate Internet connectivity into everyday objects like home appliances, medical instruments, cars and much more.

According to research done by Cisco’s Internet Business Solutions Group an estimated 25 billion ”things” are connected to the Internet. By 2020 that number could double to 50 billion.[[6]](#footnote-6) Additionally, Internet.org, an initiative that is bringing Internet access to the third world, says that only one third of the world’s population is connected to the Internet (Internet.org 2013). This is a big potential for growth for Internet companies and the Internet of Things as a whole likewise.

[[7]](#footnote-7)

Adding a sensor, processing power and connectivity to an object is enough to create an Internet of Things device[[8]](#footnote-8). A sensor to read data from the real world, a processor to process this data, which is then being sent off via a means of communication, whether that’s Wi-Fi, a mobile data connection or Bluetooth.

### The Web of Things

…

* Spark.io
* IPV6

## Business Process Modeling

…

(mention swimlanes)

## End-User Development

End-user development is the research concerned with ...

## Change Patterns

…

# State of the Art

* Yahoo Pipes, Ifttt
* Research paper Pedro/Victoria

# Research Context

## SITAC

SITAC is an international initiative to …

Key objectives of the SITAC initiative are:

1. „Facilitating seamless connection and cooperation among devices and users through the use of social networks and crowd-based applications;
2. Allowing casual users to take control of such massively deployed objects in a convenient and safe manner;
3. Providing a platform which enables the development of Social IoT and crowd- based applications and its relevant business-wise ecosystem;
4. Enabling in-node content analysis and decision making to decrease the amount of data flows;
5. Addressing technical challenges related to data analysis and recommendation techniques when leveraging the social network and crowd-based paradigms.“[[9]](#footnote-9)

Source: (ITEA 2 2013)

Objective 1 and 2 touch the aim of this work. The general focus of the Universitat Politècnica de València lies on the service layer module, which interfaces with the end user. While other parts of the system, like the actual access to hardware are developed by organizations across Europe.

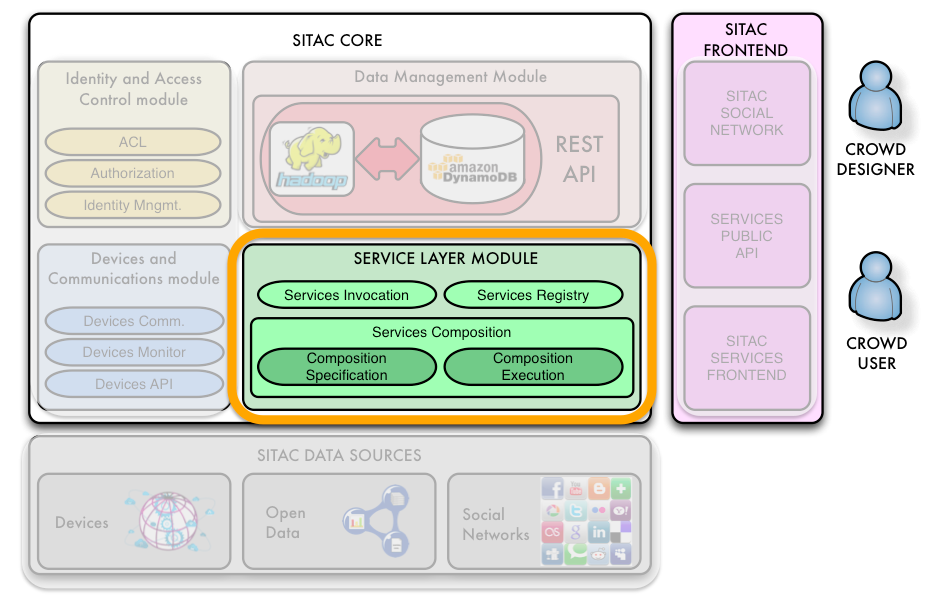


Figure 2: SITAC system architecture

## Service Layer Module

The Service Layer Module is split into 3 key areas - Services Registry, Services Composition and Services Invocation. Services Registry is a database of available services

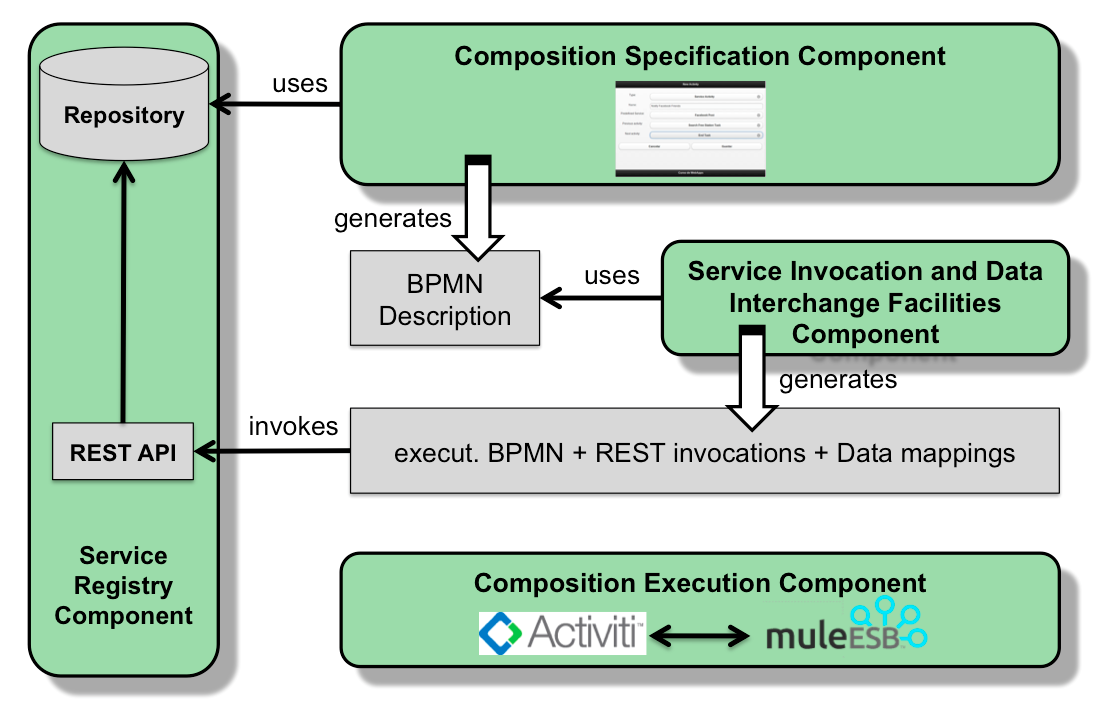


Figure 3: Service Layer Module architecture

### Services Invocation

Service invocation deals with the execution of service compositions. It …

### Services Registry

…

### Services Composition

…

# Analysis and Design

## Domain Specific Language

### Abstract Syntax

There are three times of elements in a service composition: activities, fragments and compositions.

**Activities** are remote services that either provide data or provide an action. These can be either physical devices or Internet services.

Physical devices can include sensors, like temperature sensors, pressure sensors or light sensors, and actuator, like light switches, servomotors or radiator regulation. While sensors only read data, actuators have a direct influence on the physical world. Internet services can serve as information sources or as a means of communication. These could be university services (i.e. library), a weather service, reporting the weather condition at a certain location, or messaging services, social networks (like Facebook or Twitter) and e-mail.

Activities can either have output values (like temperature reporting) or might require input data (like the location of the user). Input data can be either provided by the user or through other activities.



Figure 4: BPMN Activity

Activity: A generic Java Service Task http://www.activiti.org/userguide/

**Fragments** are conditional elements, which execute certain activities based on certain preconditions. This could be based on a weather situation, date or time, a reading by a sensor and so on. While in BPMN fragments can be nested, in this work infinite nesting is disabled for usability reasons. Compositions might get to complex for users to grasp. However, service compositions can be nested to enable this.

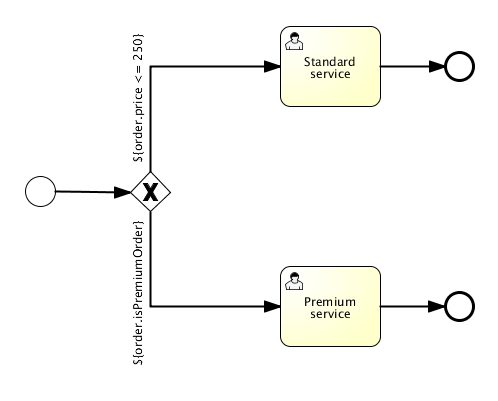


Figure 5: BPMN conditional fragment

Conditional Fragment: In a purchase the standard service or premium service is provided based on the value of the ordered items

A special case of a fragment are **parallel fragments**. These can execute multiple activities in parallel. This can be useful to speed up service execution. While one flow might be waiting on one activity to execute, other activities in the other flow can already be executed.

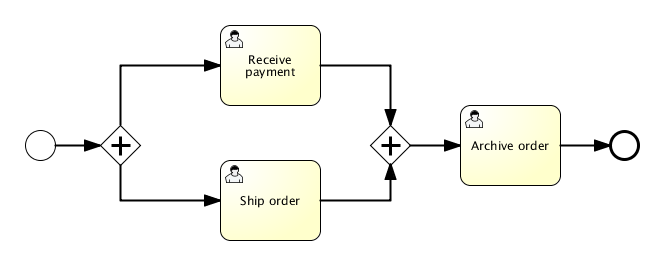
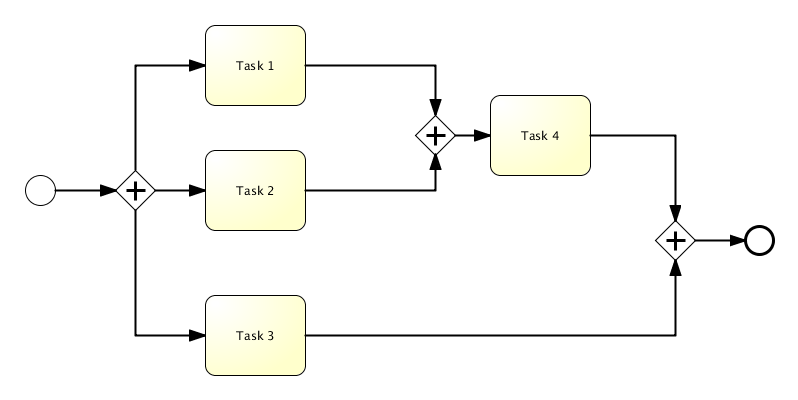


Figure 6: BPMN parallel fragment

Parallel Fragment: “Receive payment” and “Ship ordered” are executed at the same time

By combining activities and fragments into a sequence, a **service composition** is created. In BPMN terms these are called “sequence flow”. Service compositions can be nested to create more complex compositions. This means, a user might create a service composition “homecoming”, which unlocks his house’s front door, turn on the lights and plays music. This composition can be saved and integrated in other compositions that might trigger “homecoming” based on the user’s location or a certain time of the day.

Service compositions include a certain starting and end point



Service composition

The end product of this work …

* Reusability
* Compositions can be nested

Change patterns (might not have been used)

When no composition is loaded, the interface shows the tutorial, together with a clear instruction to either load a previously created composition or to create a new one. This tutorial can be pulled up at any time by the user be tapping the question-mark icon in the main toolbar.

Wiring up data:

* When no inputs are available, simple textfield is rendered

### Concrete Syntax

A first implementation of the work has been done using components from Twitter’s Bootstrap user interface framework. While BPMN

After testing this layout with users, a couple of problems where revealed:

* It wasn’t clear that the interface represented an activity flow. Users saw it rather as a list of available options than elements that are executed one after the other
* The interface looked to technical

From the technical side the decision to use Bootstrap enabled quick prototyping and iteration. However, custom interface elements and functionality that wasn’t provided by the framework was harder to implement - a trade-off that is being made by using opinionated frameworks. That’s why after the prototyping-phase a second version of the application without Bootstrap was built.

Even though in BPMN “generally, the main process is aligned horizontally” (Lucidchart 2014), a vertical layout have been chosen. Reason for this is that smartphones are typically held in portrait. This means more vertical space is available and scrolling vertically comes in more naturally.

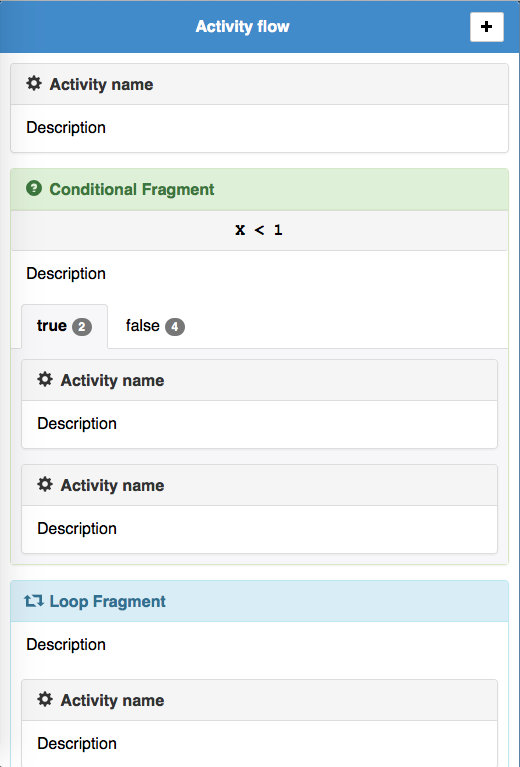
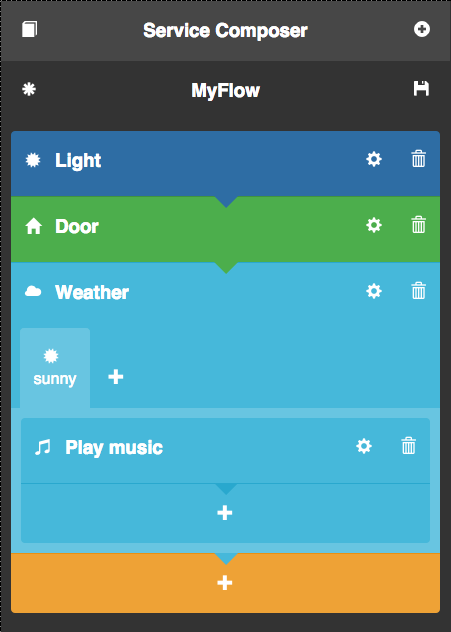
 

Figure 7: First design draft and final design

Activities and fragments contain icons that make it easier to perceive its purpose. This also counts for compositions. Users have the choice to personalize their compositions. Tapping the icon next to the compositions title reveals an icon-picker. PICTURE in the appendix.

To visualize the execution direction of the activities and provide a sense of “flow”, little arrows below flow elements point downwards.

Elements have different colors to provide a contrast to other elements. The colors are picked from a predefined color scheme and not element-specific. In the future certain activities or fragments might have specific colors that represent them.

#### Usability

The following outlines a few measures that have been taken to improve the usability of this work.

##### Buttons

Touch interfaces require close attention to the size of intractable elements of the page – namely buttons. A common button style has been used across the interface that expands every button to a minimum size of 56px by 60px – based on Google’s recommendation to size touch targets at least 48px by 48px (Google 2011). Additionally, buttons change its color on tap to give feedback to the user. On the technical side this is achieved through Webkit’s new “-webkit-tap-highlight-color” CSS property.

##### Icons

To illustrate the purpose of important user interface elements icons are used. It helps the user to recognize buttons and headlines faster and saves space that labels would take up. While for buttons an icon with a label would maximize usability, the limited screen estate and the reduction of interface clutter justified omitting the label. Research suggests that after a short learning period icon-only interfaces perform equally good as icon-and-label-interfaces (Sanchez 2012).

Figure 7: Glyphicons

The icon-set used in this work is Glyphicons[[10]](#footnote-10) that comes with Twitter’s Bootstrap interface library.

##### Warnings and error messages

To fulfill the goal of correctness-by-construction the user is guided throughout the creation process. In the upper example of an empty composition, the composition might not be incorrect. However, since it doesn’t provide any utility, a warning is shown. Warning messages are always formulated in a way that tells the user what is wrong and it also provides feedback on how to fix it. They are polite and don’t blame the user.

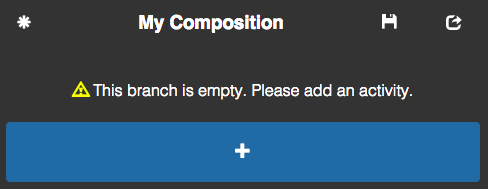


Figure 9: Warning

In cases where a user action create a defect or an incomplete composition – for instance, if a user deletes all branches of a conditional fragment – the user’s work isn’t being preserved while noticing, what’s wrong.

This procedure is in line with Jakob Nielsen’s guidelines for error messages (Nielsen 2001).

# Implementation

All of the front-end application logic is implemented in JavaScript and, therefore, runs in the user’s web browser. The server deals with data storage and supply as well as the export of a valid BPMN.

Separation of concerns is a central goal of the technical part of this work. This goes for the relationship between the server and the front-end as well as the architecture of the front-end itself. It makes components of the system independent from each other and reusable for future projects.

This work relies on web standards and can therefore, theoretically, run on any modern mobile operating system across multiple brothers. However, in praxis, special optimizations have to be made to support the same functionality across different devices. This is due to different form factors and resolutions, varying processor speeds and the nature of the standardization process, where browser vendors implement new features at different times.

Therefore, I will concentrate on Apple’s iOS with its Safari browser and Google’s Android with its Chrome browser. Safari’s browser engine is WebKit and Chrome’s is Blink, which has its roots in WebKit. This means they share the same baseline of functionality, which makes it easier to support a similar feature set among these different platforms. While this is the focus, it still doesn’t mean this web app can’t run on other platforms with other browsers as well. Sticking to web standards allows for maximum cross-platform compatibility.

## The Frontend

### Frontend logic with JavaScript

The basis of the front-end is React.js, a JavaScript library developed by Facebook, which provides functionality to responsive and modular web interfaces. The following explains how it is different from traditional approaches to front-end development and its advantages over other front-end libraries like jQuery.

#### React.js

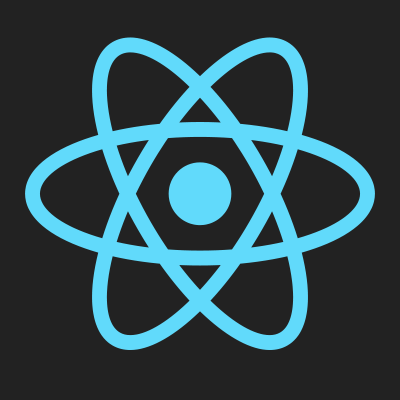
Traditionally thedeveloper needed to keep the front-end up-to-date. Every time the interface changes (through new data or user interaction), changed elements have to be updated manually by fetching it and updating its contents. The more of these atomic changes to the layout are made with jQuery the harder it gets to reason about the state of the website. It also lowers the confidence about whether the front-end representation matches the underlying data, since on complex sites it is easy to forget elements that need to be updates.

Figure 8: React.js logo

React.js automates this task. It omits the use of jQuery as the main way of manipulating the web page. In React.js the interface is always in sync with the data.

It is based on a virtual implementation of the DOM (Document Object Model), the browser’s interface to manipulate and style elements displayed on a website. To minimize the amount of DOM manipulation needed to transition the website from one state to the next one, React.js calculates the differences between both states and only applies necessary changes to the current layout. This makes it robust and increases rendering performance by avoiding common layout problems in single-page web applications, e.g. layout thrashing[[11]](#footnote-11).

React.js also helps with a clear separation of concerns. Logical entities on the webpage are separated in so-called React-components. They implement the representation as well as the presentation logic of the different user interface building blocks of the web application. Components can therefore be reused in multiple locations on the site or even in other projects. While mixing HTML and JavaScript within one component might traditionally be seen as the opposite of separating concerns, it is indeed still compatible with this concept, because strictly separating markup from its logic is rather a separation of technologies than a separation of concerns.

As this work grows and adopts new features future iterations could additionally profit from the choice of React.js as its rendering system, since React is also able to run on the server-side (Wells 2015) and wouldn’t require a rewrite of the rendering logic.

Since React handles all elements rendered to the page, injection of malicious code into the website via cross-site-scripting is ineffective. All JavaScript documents in this project make use of JavaScript’s *strict* option. Prepending ‘use strict’ to JavaScript documents results in stricter parsing by the JavaScript runtime and will throw errors in cases, where code might contain unsafe operations. For instance, when global variables are declared.[[12]](#footnote-12)

PICTURE shows the structure of this application.

#### Unidirectional data-flow with the Flux programming pattern

The programming pattern used in the front-end is called Flux[[13]](#footnote-13). It relies on a unidirectional data-flow, which means that data can only flow down a certain path, but not back. There is a single entry-point for new data.

Figure 9: Flux logo

Flux relies on three components: the dispatcher, stores and views. The dispatcher is the central entry-point for user interaction and incoming data – so called “actions”. Through a sub/pub (subscribe/publish) system stores can register callbacks with the dispatcher. When a new action comes in the dispatcher emits an event and the relevant stores update their data based on the registered callback. Views are subscribed with relevant stores and update automatically when these change.



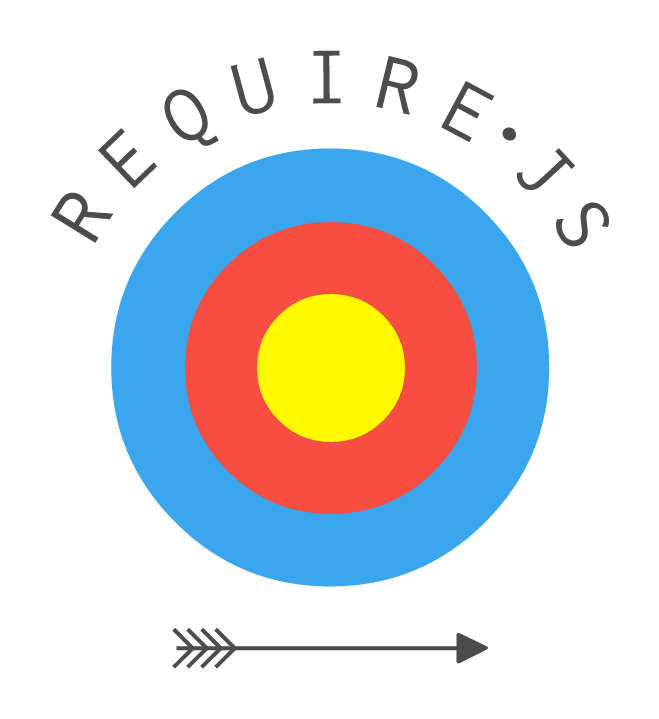
Figure 12: Flux data flow

This setup ensures that stores handle their own data, views stay up to date and that the flow of data stays replicable. The following example of deleting an activity illustrates the implementation of the Flux programming pattern.

Figure 13: Data flow example

Firstly, a user interaction is registered. In this case the user chooses to delete an activity in his composition. The corresponding React component (Activity.react) emits the “DELETE\_ACTIVITY” event. Each event carries a payload with extra information like, in this example, the activity ID. Previously, the FlowStore registered a callback with the dispatcher. When the event is emitted the dispatcher executes this callback, which results in the function invocation in the FlowStore. It deletes the activity out of the flow and emits an event that broadcasts that data has changed. As a result *every* component that relies on data from the FlowStore re-renders, bringing the interface up-to-date.

#### Dependency management with Require.js and Bower

Each React component is a separate file and Require.js[[14]](#footnote-14) is used for dependency management. Sharing functionality across multiple modules isn’t done through a global scope like jQuery (“$” variable). Using Require.js makes it possible to declare the dependencies of each module while loading these dependencies into the local scope of the module. This avoids a pollution of the global namespace and prevents conflicts.

External dependencies are loaded through Bower[[15]](#footnote-15), a package manager for web frameworks and libraries. It ensures that external content stays up-to-date.

Figure 11: Require.js logo

### Layout and Design

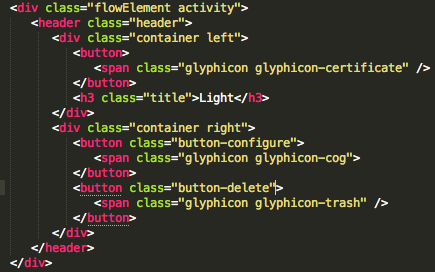
#### HTML

Since the HTML markup is rendered through React, all interface elements are implemented in JavaScript. However, the result has to be valid HTML5. While it is possible to manually render HTML elements to the DOM through React, the easier approach is to use JSX[[16]](#footnote-16). JSX is React’s own JavaScript syntax extension. It makes it possible to write HTML-like XML code within JavaScript, which gets later (using Grunt.js) transformed to real HTML. Furthermore, other React components are easily integrated, since they are available as common elements in the JSX syntax. An activity for instance can easily be displayed in the flow via <Activity data={activityData } />. What is rendered in the frontend is the result of the activity’s *render*-function.

JSX:

Macintosh HD:Users:jonashartweg:Desktop:Bildschirmfoto 2015-01-21 um 14.51.36.png

HTML:



#### CSS3 and Responsive Design

Crucial for working with mobile devices is setting the correct viewport. Usually, website are optimized for desktop browsers and therefore much bigger than the mobile screen. To make desktop websites usable smartphone browsers scale down websites to fit the screen and tell the website to render against a screen size of, i.e. in the case of an iPhone 980px[[17]](#footnote-17). In the case of mobile websites this optimization isn’t necessary. The following HTML meta-tag tells the mobile browser to set the viewport of the website to the actual device-width and to not scale the website.

Macintosh HD:Users:jonashartweg:Desktop:Bildschirmfoto 2015-01-13 um 16.12.54.png

The last bit of this tag, disallowing user scaling, has another advantage. Usually, click events by the users are delayed by roughly 300ms. This is the browser waiting, whether a double-tap is being performed – the usual gesture to zoom into a website. Disallowing user scaling disables this delay and makes the website feel more responsive (Buckler 2014).

The usage of CSS3 makes it possible to create sophisticated visual effects without the usage of images. This counts for rounded corners, drop shadows and gradients. It also offers animation functionality – a task, traditionally done via JavaScript.

CSS3 offers a new box model to layout elements on the site. Traditionally, when setting the size of an element, the border of that element is added on top of the set size. For instance, a div-container with 200px width, 10px padding and 5px border-width would result in an actual rendered size of 230px (5px left border + 10px left padding + 200px content size + 10px right padding + 5px right border). However, this makes it difficult to lay out elements on the site, since any change to the border or padding results in a change in layout. To circumvent this problem CSS3 introduces a new box model called “border-box”. Borders and paddings don’t count on top of the elements size anymore and the set width of an element matches the actual rendered with in the DOM. That’s why box model is used across the site.

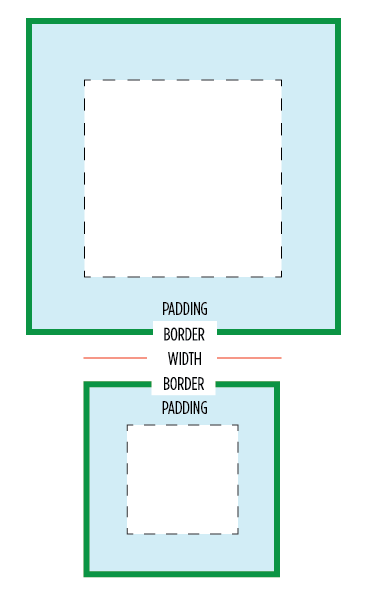


Figure 15: Comparison of content-box and border-box CSS box models

[[18]](#footnote-18)

To account for the vast number of different screen sizes of mobile devices the layout is responsive. All width declarations of elements are set in percentages instead of pixels. The layout adapts based on the current screen width.

### Building a web app

Having a website behave in a way that feels like a native application poses some challenges. With the advent of HTML5 however, these differences can party be eliminated.

The following is a non-exhaustive list of difference between a native application and a traditional website.

* Performance
  + Native application have access to hardware acceleration
  + Native applications are partly pre-compiled, while JavaScript code is interpreted just in time
* Access to hardware functionality
  + Native applications can access the file system, notifications, camera, microphone, accelerometer, hardware buttons etc.
* Offline access
  + Native applications are installed locally and run offline, websites require an internet connection

The last years brought a few of these application-exclusive features to the mobile web. Standardization work done by the W3C, the consortium that maintains web standards, and its implementation by browser vendors make it possible to build web applications that are disguised as native apps - mainly due to the way they behave and look. A website becomes a web app.

Relevant HTML5 features to this work and its application are

* AppCache: Providing offline access. This is used to make the application available when no internet connection is available
* Local-storage: A persisted store that can be used for JSON. This is used to save the user’s service compositions on the device
* CSS3 transitions: Hardware-accelerated animations. Used for all sliding elements like the MyActivites picker

Chrome’s WebView[[19]](#footnote-19) enables further integration into the mobile operating system. Interface elements like the URL bar and the browser’s control elements are hidden and a homescreen shortcut to the web app can be created.

### Development workflow

The development workflow is fully automated through Grunt.js[[20]](#footnote-20). Grunt.js is a task runner that, based on its configuration, runs certain reoccurring jobs that are usually performed by the developer. It is developed by the open-source community and has a plugin system with a big directory of available task-plugins. These include support for LESS compilation, prefixing CSS, copying and cleaning directories and a static web server with live reloading functionality.

Figure 13: Grunt logo

The setup used in the project is based on “generator-gosub”[[21]](#footnote-21), a development boilerplate developed by me for gosub communications GmbH.

Below is a list of tasks used in this configuration. It’s split in tasks that are run during the development and tasks that are used to create a production-ready version of the website. Development tasks are run automatically when project files change. Production tasks are run on demand.

Development tasks:

* React: JSX syntax in the JavaScript source-files is transformed into plain JavaScript.
* LESS: LESS files are transformed into CSS. Source-maps are generated to support LESS debugging in the browser’s development tools.

Production tasks:

* Require.js: Based on the Require.js configuration all JavaScript source-code is concatenated into one file. This reduces HTTP requests made to the server when the website is hosted and leads to faster loading speed.
* Auto-prefixer: CSS files are processed to contain relevant vendor-prefixes. While certain CSS functions are in the process of standardization browser vendors already implement them. However, hidden under a vendor-specific prefix like “-moz-“ for Mozilla Firefox or “-webkit-“ for WebKit based browsers.
* File operations: All relevant files are exported to its own distribution directory. Excluded files are, for instance, LESS styles.

The development workflow is documented in the “readme.md” file in the root directory. It contains an explanation of Grunt.js, all used development packages and an installation guide to setup the project. The document is adapted from the aforementioned “gosub-generator” project. It is written in Markdown[[22]](#footnote-22), a markup language that easily can be converted to HTML. The documentation can be found in the appendix.

### Software stack and debugging

GitHub’s open-source text editor Atom[[23]](#footnote-23) is used to write code. The source code is hosted in a private repository on GitHub. Google’s Chrome browser is used for testing purposes.



Figure 17: Chrome, Atom and GitHub logo

The central tool to debug the application are the Chrome DevTools[[24]](#footnote-24) - a toolkit, integrated in Google Chrome. It offers insight into the DOM, applied styles, current memory usage, network-communication and rendering performance. It also offer just-in-time manipulation of the website to quickly test different layout configurations and style changes.

A key feature of the DevTools is remote debugging, which offers a way to access the mobile browser and to debug the application on the target device itself. It also uses USB port forwarding to route the web page on the development machine to the smartphone. This makes the use of a local development web server unnecessary.

PICTURE shows the complete setup with the testing device on the left side and the DevTools on the right side.

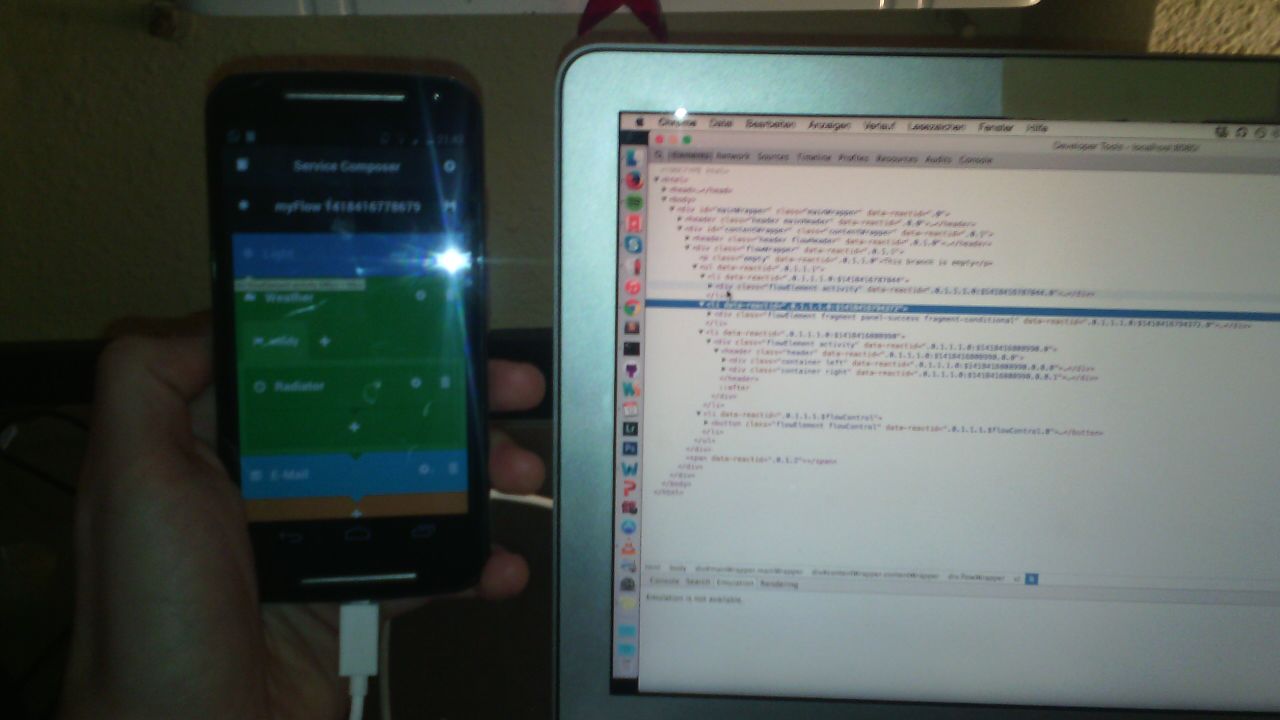
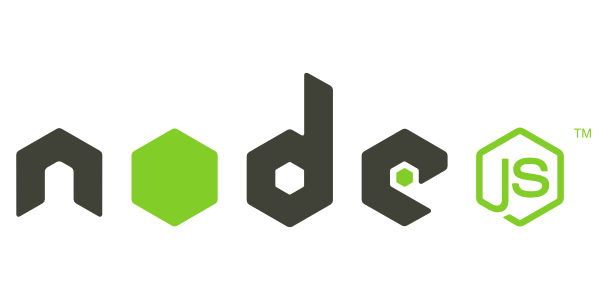


Figure 18: Working with remote debugging

## The Backend and BPMN generation

The transformation of the composition from JSON to BPMN is done on the server-side. Node.js is used as a backend.

#### Node.js and Express

Node.js[[25]](#footnote-25) is a JavaScript webserver. It is based on Google’s V8 JavaScript engine and executes its code in a non-blocking fashion.

This concept is known as “asynchronous I/O”, which states that pending operations (like file-read or network requests) don’t block the further execution of a thread. Synchronous applications stop a thread until a pending read or write action to the file system or the network terminates and then proceeds with its execution. This can be inefficient, since this idling time could be used to execute other tasks that don’t rely on the requested data.

Asynchronous applications, however, never stop the execution of threads, which results in faster response times.

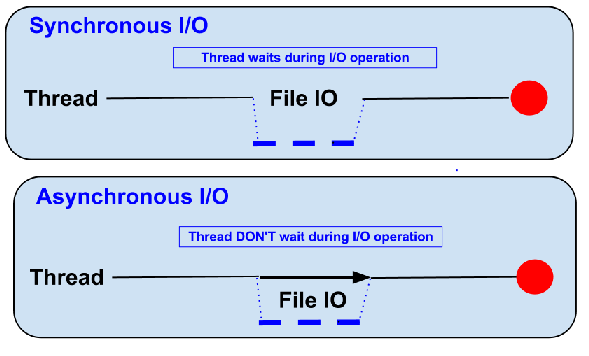


Figure 19: Synchronous and Asynchronous I/O

[[26]](#footnote-26)

This project profits from the fact that the same programming language is used in the frontend and backend - JavaScript in the browser and JavaScript on the server. This makes the work with JSON data easier, since it is native to JavaScript. It also enables flexible decisions whether certain tasks should be executed on the client or the server.

For instance, in early prototypes of this work the transformation of JSON to BPMN was done on the client. However, since the execution of the BPMN happens on the server anyways, shifting this functionality to the server made sense. This was as easy as copying the transformation code from the client to the server, since both rely on JavaScript.

This approach follows the trend of *Isomorphic JavaScript*[[27]](#footnote-27). While nowadays most of the application logic is run on the client-side it sometimes makes sense to run the same code on the server-side. It gives flexibility to run certain, performance-critical tasks on the server, to make the interface more responsive. It lets the server and client share data models for easier data synchronization. And it makes it possible to serve server-rendered versions of the website to search engine crawlers, which in theory understand JavaScript, but usually struggles to interpret JavaScript-rendered web applications (Gray 2013).

Coupled with the option to run React.js components on the server, future iterations of this work could lead to an isomorphic web application.

Express.js[[28]](#footnote-28) is used to process requests made to the server.

### Data structure and conversion

Each activity contains the following information:

* ID
* Name
* Description
* Icon
* URL
* Input arguments
* Output arguments

#### JSON structure

The JSON structure is designed with the final conversion to XML in mind. Suggestions on how to model it have been taken from a 2006 blog article at xml.com on how to convert between JSON and XML (Goessner 2006). This increased compatibility between the JSON and XML representation of activity flows.

Each activity looks as follows:

…

#### BPMN structure

…

# Conclusions

…

* Database would’ve been helpful

## Limitations and Future Work

An expert mode could enable this feature along with further control over data flow and data types as well as a way to let a user create his own activities by manually linking it to web services. One enhancement to this work would also be making it possible to move elements (activities and fragments) up and down the flow. Right now this is not allowed, since it could break the composition. If activities that provide data can be moved under activities that require its data, the composition would break. The expert mode could also encompass infinite nesting of fragments. A feature that is technically already implemented, but deactivated, since it might confuse novice users. Finally, other concepts of BPMN could be implemented. One of these is the idea of “swimlanes”, where activities are associated with certain user roles in the execution of these services.

In later stages of the project, at a point where SITAC reaches a stage where all components are interlinked, the social network component of SITAC’s objectives could come in play. It is imaginable to create a platform, where users can share their compositions with other users.

An area that is untouched so far is Service Invocation - an interface for executing user compositions. This field is part of future work performed at the UPV. Early mockups of this, created by Victoria Torres Bosch, can be found in the appendix PICTURE.

# Literaturverzeichnis

Buckler, Craig. *5 Ways to Prevent the 300ms Click Delay on Mobile Devices.* 08. 01 2014. http://www.sitepoint.com/5-ways-prevent-300ms-click-delay-mobile-devices/ (Zugriff am 06. 02 2015).

Covert, Ed, und Angela Orebaugh. *Ethical challenges of the Internet of Things.* 29. 01 2014. http://www.scmagazine.com/ethical-challenges-of-the-internet-of-things/article/331460/ (Zugriff am 06. 02 2015).

Goessner, Stefan. *Converting Between XML and JSON.* 31. 05 2006. http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html (Zugriff am 06. 02 2015).

Google. *PageSpeed Insights.* 11. April 2011. https://developers.google.com/speed/docs/insights/SizeTapTargetsAppropriately (Zugriff am 06. 02 2015).

Gray, Michael. *How Google’s Javascript Bot Can Kill Your Website.* 13. 02 2013. http://graywolfseo.com/google/google-javascript-bot/ (Zugriff am 06. 02 2015).

Greenwald, Glenn , Ewen MacAskill, Laura Poitras, Spencer Ackerman, und Dominic Rushe. *The Guardian.* 12. 07 2013. http://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data (Zugriff am 06. 02 2015).

Internet.org. „Technology leaders launch partnership to make internet access available to all.“ August 2013. https://fbcdn-dragon-a.akamaihd.net/hphotos-ak-prn1/851572\_595418650524294\_1430606495\_n.pdf (Zugriff am 06. 02 2015).

ITEA 2. „Social Internet of Things: Apps by and for the Crowd .“ 12 2013. https://itea3.org/project/result/download/6699/11020-SITAC-SITAC\_profile\_dec-13.pdf (Zugriff am 06. 02 2015).

Lucidchart. *BPMN & BPMN 2.0 Tutorial.* 2014. https://www.lucidchart.com/pages/bpmn-bpmn-2.0-tutorial (Zugriff am 06. 02 2015).

Nielsen, Jakob. *Error Message Guidelines .* 24. 06 2001. http://www.nngroup.com/articles/error-message-guidelines/ (Zugriff am 06. 02 2015).

Sanchez, Edward. *Icons vs Labels vs Both.* 30. 10 2012. http://edwardsanchez.me/blog/13589712 (Zugriff am 06. 02 2015).

Valderas, Pedro, Torres, Victoria, Victoria Torres, Pedro Valderas, Vicente Pelechano Ignacio Mansanet, Ignacio Mansanet, und Vicente Pelechano. „End-User Service Compositions in the IoT.“ Valencia, 2013.

Wells, David. *Serverside React Rendering: Isomorphic JavaScript with ReactJS + Node.* 01. 01 2015. https://reactjsnews.com/isomorphic-javascript-with-react-node/ (Zugriff am 06. 02 2015).

# Table of Figures

Figure 1: Use-Case BPMN (Source: End-User Service Compositions in the IoT) 7

Figure 2: SITAC system architecture 14

Figure 3: Service Layer Module architecture 14

Figure 4: BPMN Activity 16

Figure 5: BPMN conditional fragment 17

Figure 6: BPMN parallel fragment 17

Figure 7: First design draft and final design 19

Figure 9: Warning 20

Figure 12: Flux data flow 24

Figure 15: Chrome, Atom and GitHub logo 29

Figure 16: Working with remote debugging 30

Figure 17: Synchronous and Asynchronous I/O 31

# Appendix

* Defined meta-models
* Mockups of the UI
* Documentation

1. http://tx.english-ch.com/teacher/len/level-c/the-internet-of-things-2010-conference/ [↑](#footnote-ref-1)
2. https://en.wikipedia.org/wiki/Smart\_client [↑](#footnote-ref-2)
3. https://itea3.org/project/sitac.html [↑](#footnote-ref-3)
4. https://itea3.org/ [↑](#footnote-ref-4)
5. https://en.wikipedia.org/wiki/Code\_of\_conduct [↑](#footnote-ref-5)
6. http://share.cisco.com/internet-of-things.html [↑](#footnote-ref-6)
7. http://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342 [↑](#footnote-ref-7)
8. https://www.youtube.com/watch?v=Jqmc0j7a600 [↑](#footnote-ref-8)
9. SITAC Profile (URL: (ITEA 2 2013)) [↑](#footnote-ref-9)
10. http://glyphicons.com/ [↑](#footnote-ref-10)
11. http://www.kellegous.com/j/2013/01/26/layout-performance/ [↑](#footnote-ref-11)
12. https://msdn.microsoft.com/en-us/library/ie/br230269%28v=vs.94%29.aspx [↑](#footnote-ref-12)
13. http://facebook.github.io/flux/ [↑](#footnote-ref-13)
14. http://www.requirejs.org/ [↑](#footnote-ref-14)
15. http://bower.io/ [↑](#footnote-ref-15)
16. https://facebook.github.io/react/docs/jsx-in-depth.html [↑](#footnote-ref-16)
17. https://developer.apple.com/library/ios/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html [↑](#footnote-ref-17)
18. http://css-tricks.com/box-sizing/ [↑](#footnote-ref-18)
19. https://developer.chrome.com/multidevice/webview/overview [↑](#footnote-ref-19)
20. http://gruntjs.org/ [↑](#footnote-ref-20)
21. https://github.com/gosub-communications/generator-gosub [↑](#footnote-ref-21)
22. https://daringfireball.net/projects/markdown/ [↑](#footnote-ref-22)
23. https://atom.io/ [↑](#footnote-ref-23)
24. https://developer.chrome.com/devtools [↑](#footnote-ref-24)
25. http://nodejs.org/ [↑](#footnote-ref-25)
26. http://www.techthali.org/wp-content/uploads/2012/07/npm8.png [↑](#footnote-ref-26)
27. http://nerds.airbnb.com/isomorphic-javascript-future-web-apps/ [↑](#footnote-ref-27)
28. http://expressjs.com/ [↑](#footnote-ref-28)