Rule Composer

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2018

Autor(en):	Roman Lacher
Betreuer:	Daniel Politze
Projektpartner:	Siemens AG Zug
Projektpartner	Markus Winterholer
Projektpartner:	Hermann Mayer

Inhaltsverzeichnis

Abbildungsverzeichnis

1. AUFGABENSTELLUNG

Die Platzierung von Sensoren in Gebäuden erfolgt anhand von Regelwerken, welche für den ordentlichen Betrieb und die Abnahme durch Behörden befolgt werden müssen. Fehler in der Platzierung erfordern kostspielige Umbaumaßnahmen und führen zu Verzögerungen im Bau.
Digitale Modelle werden verstärkt zur Planung, Simulation und Projektabwicklung eingesetzt. Digitale Modelle, oder digitale Zwillinge (Digital Twins) von Gebäuden (Building Information Model BIM) ermöglichen die automatische Platzierung von technischen Geräten anhand der im Modell enthaltenen Dimensionen und der Angaben zur Raumnutzung. Digitale Modelle der technischen Geräte können somit im virtuellen Raum platziert werden. Die vollständige digitale Modellierung aller Gewerke in einem Gebäude zusammen mit der Ausstattung und den Positionsdaten der sich im Gebäude befindenden Personen, ermöglicht eine Simulation aller Abläufe in einem Gebäude.
Als Beispiel für die sich aus der Digitalisierung eines Gebäudes ergebenden Möglichkeiten, soll die automatisierte Platzierung von Brandmeldeanlagen in einem Gebäudemodell betrachtet werden. Dabei liegen bereits digitale Modelle des Gebäudes und der zu platzierenden Anlagen vor. Das Regelwerk zur Platzierung besteht aus technischen Regeln aufgrund der Dimension und des Arbeits- und Installationsraumes und aus gesetzlichen Bestimmungen, die bei der Positionierung berücksichtigt werden müssen. Eine Konfliktvermeidung mit anderen Gegenständen fließt in die Positionierung mit ein. Um die Platzierung zu automatisieren muss ein Algorithmus die verschiedenen Regeln interpretieren, um eine mögliche Position des Geräts im Raum vorzunehmen
Die Regeln für die Platzierung von Sensoren werden heute in XML Dateien beschrieben. Um die Beschreibung neuer Regeln und die Änderung vorhandener Regeln für den Anwender möglichst einfach zu gestalten, soll ein grafischer Regeleditor implementiert werden, welche die Bearbeitung von Regeln ermöglicht. Der Editor soll in eine bestehende Amazon Web Service Cloud Computing Plattform integriert werden.

<u>Aufgaben</u>:
Analyse bestehender Strukturen für Regeln.
Implementierung eines grafischen Regeleditors zur Eingabe von Regeln.
Installation des Editors in der Amazon Web Services Plattform

2. ABSTRACT

In der heutigen, modernen Welt wird häufig mit digitalen Kopien von realen Objekten gearbeitet. Oft werden diese Kopien bereits vor den echten Originalen in der Planungsphase geschaffen. Im Gebäudebau wird vielfach noch traditionell gearbeitet. Die Firma Siemens AG möchte dies ändern. Mit dem Konzept eines „digital Twins“ haben sie bereits in anderen Bereichen Erfolg. Das Ziel ist es dabei, in der Planungs- und Bauphase die Kommunikation aller beteiligten Personen (wie z. B. Architekten, Generalunternehmer) zu verbessern. Gerade bei Gebäudetechnik ist nach dem eigentlichen Bauen die Wartung sehr wichtig. Dabei hilft so ein Modell beträchtlich. Ziel dieser Studienarbeit ist es, die Verteilung von Gebäudetechnik, wie zum Beispiel Feuermelder, zu automatisieren. Dabei gibt es für viele Länder verschiedene Regeln und Normen, welche beachtet werden müssen. Mit dem Regeleditor können diese Regeln verwaltet werden. Der „digital Twin“ kann in einer Webansicht betrachtet werden. In dieser Ansicht kann man ein Raum als „Space“ aussuchen, der als Basis für die Verteilung dient. Danach können mithilfe der Regeln, die Feuermelder auf alle Räume mit dem gleichen „Muster“ automatisch platziert werden. Dies bringt dem Gebäudetechniker, welcher diese Feuermelder planen muss, eine enorme Zeitersparnis. Ausserdem wird dadurch die Einhaltung der Regeln sichergestellt.

3. MANAGEMENT SUMMARY

3.1 AUSGANGSLAGE

Mit der „Industrie 4.0“ bezeichnet man die Idee, industrielle Produktion weiter zu verbessern. In der Bauplanung ist dies noch nicht sehr weit fortgeschritten. Der Grund ist, dass das Bauwesen relativ gut funktioniert und bisher kein Bedarf bestand. Seit den „Internet of Things“ und den „Smart Home“ hat sich dies aber geändert. Sämtliche Gebäudetechnik muss heutzutage zusätzlich geplant werden. Da es immer mehr Technik zu berücksichtigen gibt, bekommen Generalunternehmer immer mehr Zeitprobleme. Diesen zeitaufwändigen Prozess möchte man mindern, indem die Gebäudetechnik automatisch platziert wird.
Mit BIM (Building Information Management) wurde die Digitalisierung der Bauwirtschaft initiiert. Auf dem Markt gibt es einige Softwareprodukte für BIM, darunter Revit von Autodesk oder ArchidCAD von Graphisoft. Mit der Absprache von der Firma Siemens AG wird in dieser Arbeit das Open Source Programm xBIM welches ein Projekt von Steve Lockley von der Northumbria University ist, verwendet. Da der Rule Composer als Webanwendung laufen soll, wurde xBIM WebUI ausgewählt, welches eine asp.net Webanwendung von xBIM ist.
In meiner Arbeit geht es spezifisch um Feuermelder, welche automatisch platziert werden sollen, jedoch sollen später beliebige Objekte in beliebigen Räumen verteilt werden können. Dafür müssen alle Parameter für die Verteilung bestimmen werden. Diese Parameter sind die Hauptaufgabe dieser Arbeit. Aus diesen Parametern wird ein Algorithmus die Verteilung vornehmen. Der Algorithmus selber ist nicht Teil dieser Arbeit.

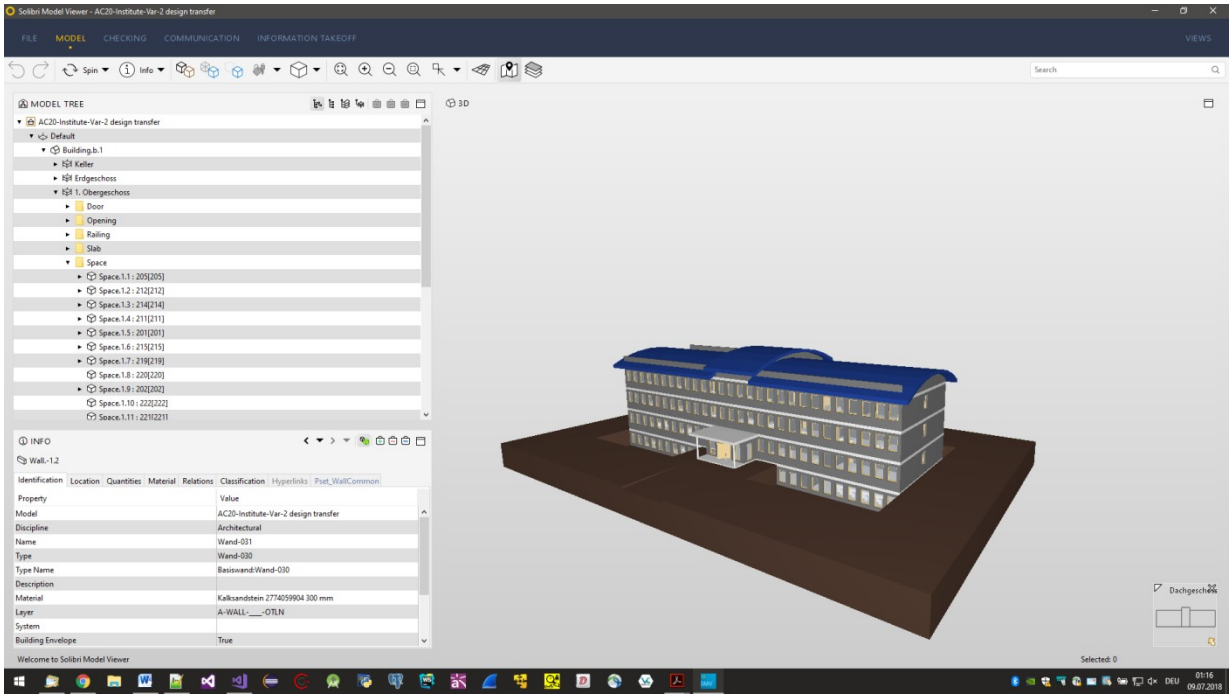

Abbildung 1 Dreidimensionale BIM Ansicht von Solibri Model Viewer

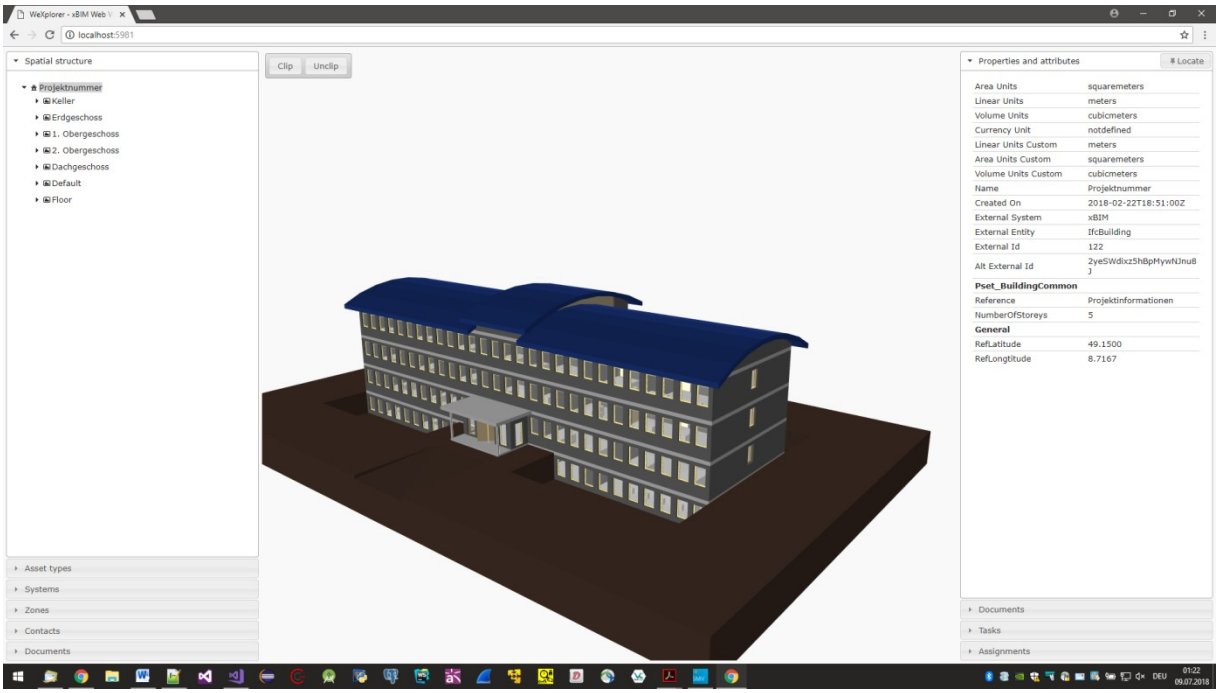

Abbildung 2 Dreidimensionale BIM Ansicht von xBIM WebUI

3.2 Vorgehen und Technologien

In einem ersten Schritt musste ich mich in BIM einarbeiten. BIM ist eine für Gebäude angepasste CAD Variante. Das Datenformat für BIM ist IFC. In einem IFC werden alle Objekte (Wände, Flächen, Türen, Möbel, Gebäudetechnik) zusammen mit ihren ID Nummern und anderen Attributen aufgelistet. BIM Software können aus diesen IFC Dokumente eine dreidimensionale Ansicht erstellen.
Des Weiteren habe ich ein Beispiel Output im XML Format bekommen, welches von Hand geschrieben wurde. Der Output, welcher in dieser Arbeit generiert wird, ist hingegen im JSON Format. Der erste Prototyp wurde in TypeScript geschrieben. Leider wurde relativ schnell klar, dass TypeScript für die Arbeit ungeeignet ist, deshalb wurde das Projekt in Javascript geschrieben. Die Arbeit wurde in Absprache von Herrn Markus Winterholer und Herrn Hermann Mayer gehalten. Herr Winterholer ist der direkte Industriepartner und Auftragsgeber dieser Arbeit. Herr Mayer gab die Technischen Anforderungen an das Projekt. Der Rule Composer soll auf aws (Amazon Web Services) laufen.

3.3 Ergebnisse

Das Resultat ist eine Erweiterung für den xBIM WebUI. Aus der Webansicht eines Gebäudes kann man mithilfe des Rule Composers ein Pattern für das Platzieren aussuchen. Ausserdem ist es möglich das Regelset auswählen, welches zum Einsatz kommen soll. Bei Bedarf kann man Regelsets editieren, erstellen und löschen. Zum Schluss wird das zu platzierende Objekt eingegeben. Als Resultat

bekommt man ein JSON File, welches alle eingegebenen Parameter beinhaltet. Ursprünglich lag der Fokus auf den Regelsets, welche der Generalunternehmer definiert. Während der Arbeit stellte sich jedoch heraus, dass das Extrahieren des Pattern viel aufwändiger ist und deshalb ins Zentrum der Arbeit rückte.

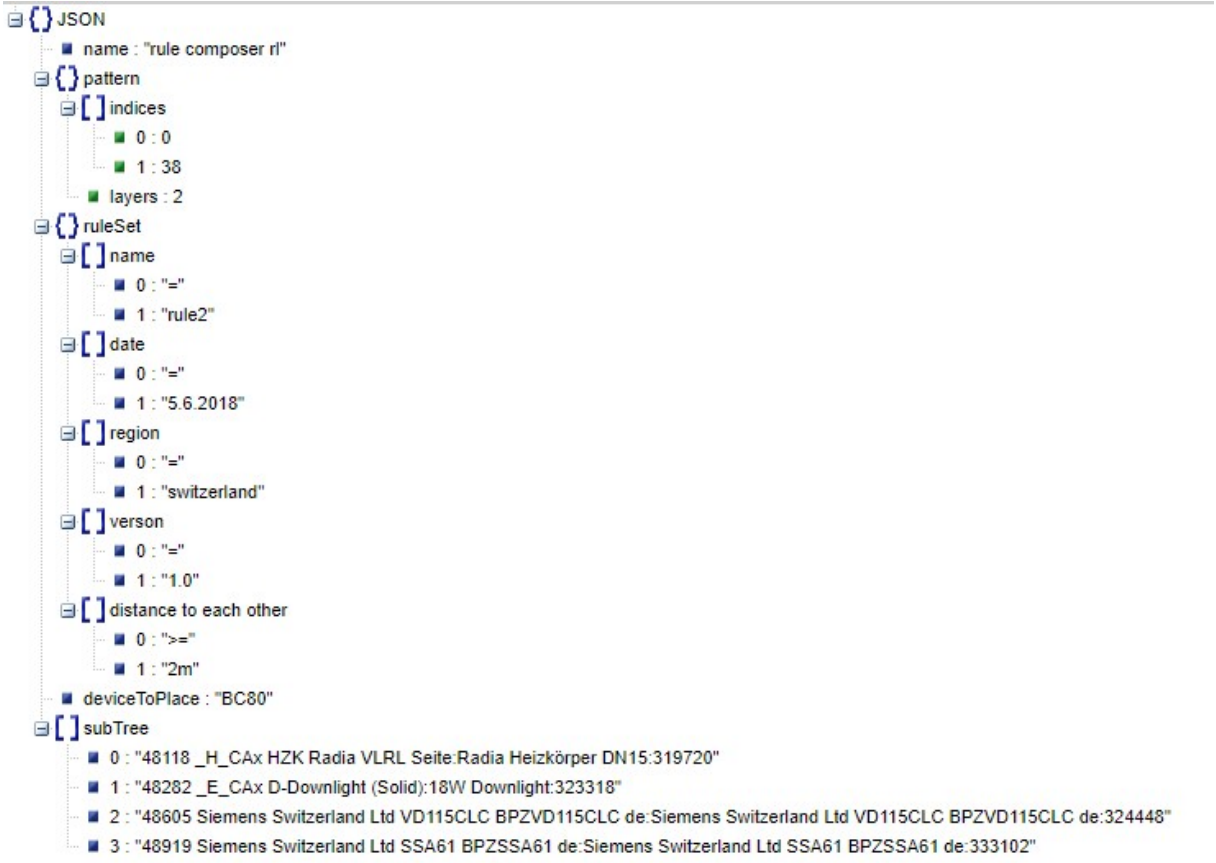

Abbildung 3 Beispiel eines Output JSON

3.4 AUSBLICK

Diese Arbeit ist der erste Schritt für das Automatisieren der Verteilung von Gebäudetechnik. In einem nächsten Schritt muss das von dieser Arbeit erstellte JSON von einem Algorithmus eingelesen werden und dann entsprechend die Objekte in den eingestellten Räumen verteilen.

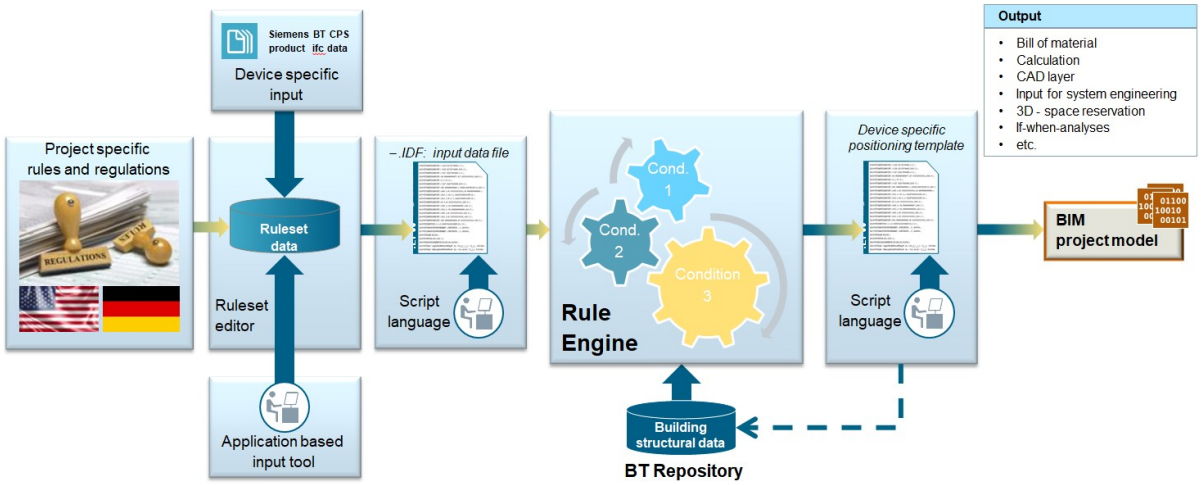

Abbildung 4 PoC Rule Composer im Gesamtbild

4. TECHNISCHER BERICHT

4.1 URSPRÜNGLICHE ARBEITSPAKETE

Folgende Arbeitspakete waren Ursprünglich geplant:

Titel	Beschreibung	Aufwand	Priorität
Ober Baum Pattern	Auslesen und Parsen des ausgewählten Ober Baum Elementes, um das Pattern für die spätere automatische Verteilung zu ermöglichen.	Sehr Hoch	Hoch
Unter Baum Pattern	Auslesen und Parsen der ausgewählten Elemente im Unterbaum. Diese können verwendet werden, um zusätzliche Muster für das automatische Verteilen anzubringen.	Hoch	Hoch
Regelsets	Regelsets müssen erstellt, bearbeitet und gelöscht werden können. Das Bearbeiten der Regeln muss möglichst frei und flexibel sein, damit auch vom Entwickler unvorhergesehene Regeln ermöglicht werden können. Trotzdem soll es benutzerfreundlich und intuitiv sein.	Hoch	Sehr Hoch
Auswahl zu platzierendes Objekt	Das zu verteilende Objekt. Es kann aus einer Auswahl selektiert werden oder ein eigenes hinzugefügt werden. Das Objekt besteht aus Name, Icon und Grösse.	Mittel	Hoch
Speichern des Output JSONs	Das Resultierende JSON soll lokal und auch in einem aws s3 bucket gespeichert werden.	Tief	Hoch
Konvertierter IFC zu wexBim und JSON	Die Webanwendung von xBim kann IFCs nicht direkt lesen, sondern braucht eine für sie optimiertes Speicherformat.	Tief	Mittel
Darstellung des 3D Modells	Bei der Ansicht der einzelnen Stockwerke und Räume die Möglichkeit geben, obere Bereiche auszublenden und Räume einzufärben.	Hoch	Tief
Installationsanleitungen	Es müssen Installationsanleitungen für die Solutions und Cloud Konfiguration geschrieben werden.	Tief	Hoch
Sicherstellen der Kompatibilität	Das JSON Resultat des RuleComposer soll auch nach dem anwenden an alle Räume und hinzufügen von extra Eigenschaften weiterhin mit xBim kopatibel bleiben	Mittel	Mittel
Persistente Datenbank für Regeln	Auf aws existiert eine Persistente Datenbank auf der alle Regelsets gespeichert sind.	Mittel	Hoch

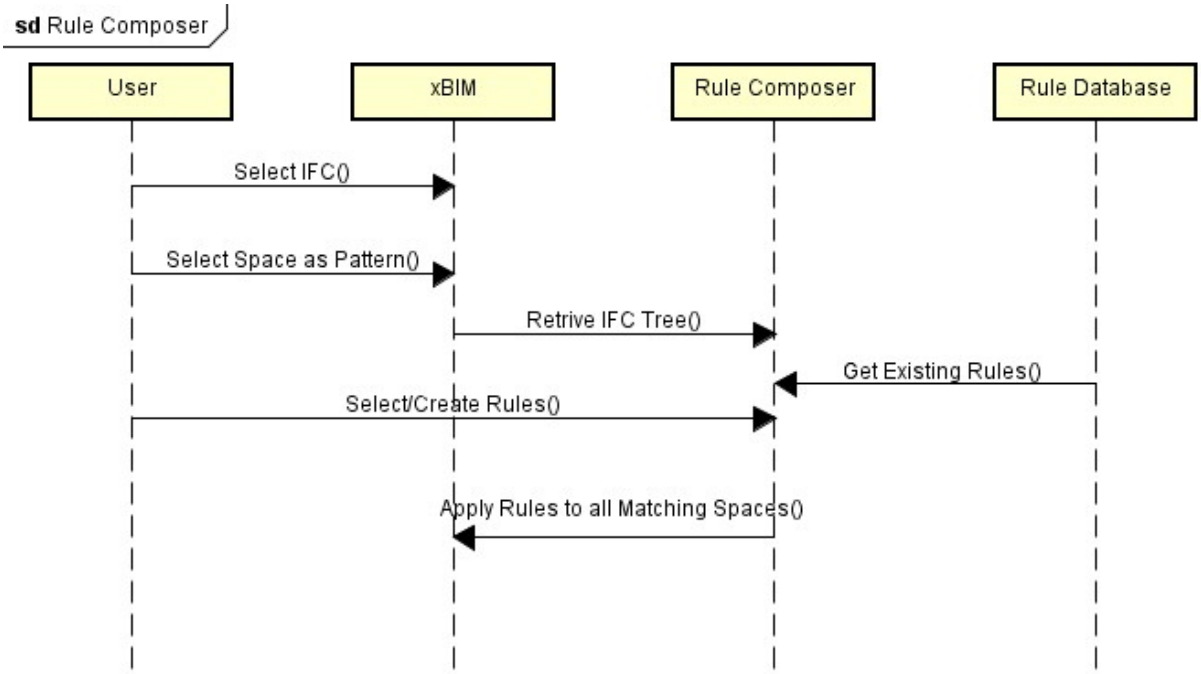

Abbildung 5 Geplantes Sequenzdiagramm des Rule Composers

4.1.1 Oberbaum Pattern

Der erste Teil für den Output des Rule Composers ist das Pattern. Dieses Pattern beschreibt das Muster im IFC Baum des vom User ausgewählten Raumes. Es wird benötigt, um später alle anderen Räume mit dem gleichen Pattern zu finden. Diese Räume kommen dann für die automatische Platzierung der Regeln in Frage.

4.1.2 Unterbaum Pattern

Zusätzlich zu dem obersten Element im IFC Baum, kommt ein Unterbaum. Dieser ist optional. Er wird benötigt, falls der User die Verteilungsregel auch abhängig von Objekten im Raum gestalten möchte. In diesem Unterbaum sieht man zum Beispiel Möbel, Türen, Fenster und andere Gebäudetechnik. Der User hat im Rule Composer freie Wahl, diesen Unterbaum zu gestalten. Je spezifischer er diesen definiert, desto genauer werden die Räume, welches das untergeordnete Pattern besitzen, ausgewählt. Beim Auswählen des Unterbaumes gibt es 2 verschiedene Implementationsmöglichkeiten. Es kann bei jedem Raum ein Regelknopf erstellt und dann in einem nächsten Schritt der Unterbaum ausgewählt und konfiguriert werden. Oder es wird ein allgemeiner Regelknopf erstellt, welcher beim Drücken anhand der derzeitigen Auswahl der Unterelemente entscheidet, wie das Unterpattern aussieht.

4.1.3 Regeln

Die Regeln selber bestimmen, wie die Objekte platziert werden müssen. Diese Regeln sind in Regelsets gespeichert, welche der User auswählen kann. Die Regelsets können erstellt und editiert werden. Diese beschreiben Abstände zu Wänden, Fenstern, Türen oder anderen Elementen. Ausserdem werden Abstände zu den zu platzierenden Elementen selber definiert. Regelsets haben alle eine Version, damit selbst bei Änderungen alte Versionen noch brauchbar bleiben. Ausserdem enthalten die Regelsets Länder, Regionen, Gebäudeart und Zonen Attribute, damit optionale Features in Zukunft hinzugefügt werden können.

4.1.4 Objekt zum Platzieren

Das Projekt ist vor allem für Feuermelder ausgelegt, jedoch können mit dem Rule Composer beliebige Objekte platziert werden. Diese Objekte haben alle zusätzliche Icons, welche später auf Baupläne ersichtlich sein müssen. Ausserdem werden genaue Gerätenamen und Spezifikationen wie Grösse und Anschlüsse definiert.

4.1.5 Darstellung des 3D Modells

Die Webansicht von xBim ist für das Auswählen des Raumes leider nicht sehr intuitiv, da alle Wände und Decken in einen bläulichen durchsichtigen Modus dargestellt werden. Eine Lösung wäre, die darüber gelegenen Stockwerke und die Decke komplett unsichtbar zu machen und die Böden jeweils einzufärben.

4.2 Ergebnisse der Arbeitspakete

Resultate der Arbeitspakete

4.2.1 Ober-/Unterbaum Pattern

Die Pattern sind ein essenzieller Teil der Arbeit. Diese Arbeitstakte werden wie geplant umgesetzt, wobei wie vorausgesehen der Oberbaum mehr Aufwand kostete. Die Schwierigkeit liegt in dem sehr dynamischen Aufbau des semantischen Baumes. Dadurch, dass die „Tiefe“ des ausgewählten Spaces nicht festgelegt ist, muss der Code eine beliebige Tiefe unterstützen. Dies wurde rekursiv gelöst und die Tiefe wurde als *pattern.layers* beim Output mitgegeben.

Eine weitere Schwierigkeit ist, dass JavaScript bei Objekte nur die Referenzen kopiert. Dies ist nichts Neues, da das ein Standartverhalten bei vielen Programmiersprachen ist. Die Lösung dafür ist *Object.create()* welche sich als unbrauchbar für meine Arbeit herausstellte, weil *Object.create()* nur die oberste Ebene des Objektes kopiert. Da der semantische Baum jedoch aus sehr vielen Ebenen besteht, muss auf *Object.assign({ }, objectToCopy)* zurückgegriffen werden, welches eine echte Kopie des Objektes mit allen Ebenen erzeugt.

```
function findElement(fullTree, id) {
    if (fullTree.id == id) {
        composer.entity = Object.assign({}, fullTree);
        composer.patternTreeFound = true;
    }
    else {
        if (fullTree.children) {
            for (var i = 0; i < fullTree.children.length; i++) {
                composer.patternTreeIndices.push(i);
                findElement(fullTree.children[i], id);
                if (composer.patternTreeFound) {
                    return;
                }
                composer.patternTreeIndices.pop();
            }
        }
    }
}
```

Abbildung 6 Beispiel einer rekursiven Funktion welche ein Element im semantischen Baum findet

4.2.2 REGELSETS

Das Verwalten der Regelsets ist möglichst flexibel gestaltet worden. Es gibt für die Regelsets nur minimale Einschränkungen. Regeln müssen immer einen Namen , Datum, Region und Version haben. Die eigentlichen Regeln sind aber völlig frei einstellbar, um die maximale Flexibilität zu erreichen. Jede Regel ist mit einem Operator ausgestattet. Dies ist ein Zusatz, der von Herrn Mayer vorgeschlagen wurde, da es das spätere Einlesen und Parsen erheblich vereinfacht (siehe Gesprächsprotokoll). Die Hauptschwierigkeit hierbei ist, dass in dem neuen Fenster ein anderer Kontext herrscht. Dies bedeutet, wenn man aus dem alten Fenster auf das neue zugreifen möchte, muss man dies über *composer.mewWindow* machen. Wenn man aus dem neuen Fenster (meist durch Events) auf den alten Kontext zugreifen möchte, muss man *window.opener* verwenden. Die Folge ist, dass man weniger Code wiederverwenden kann, da einige Funktionen nur für ihren eigenen Kontext verfügbar sind.

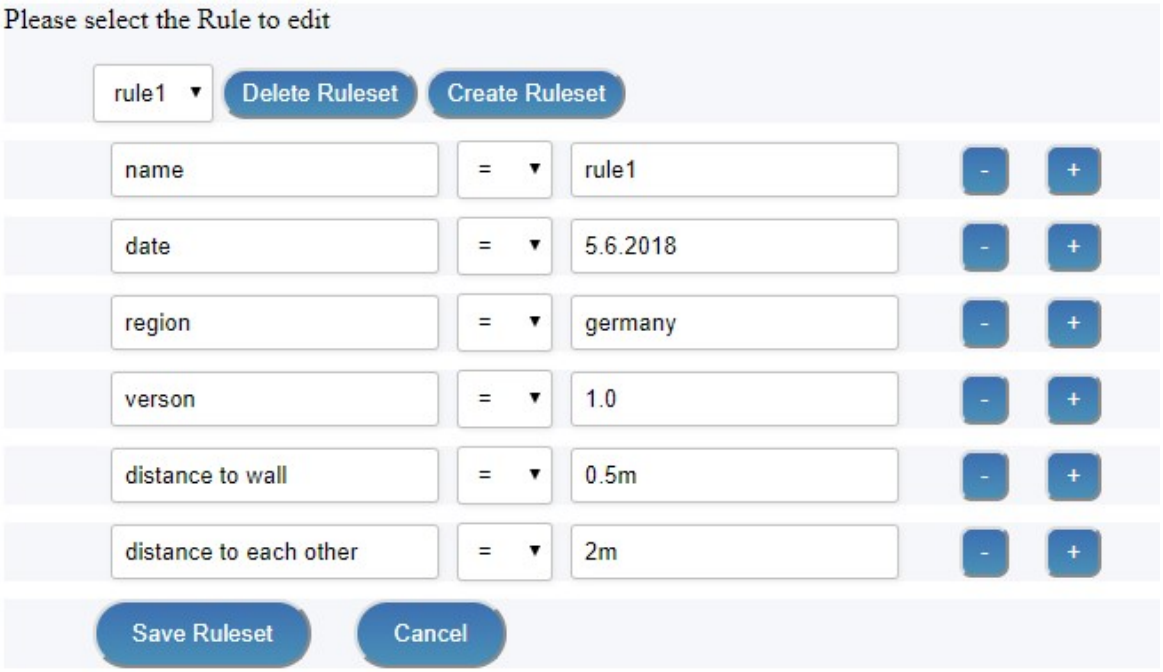

Abbildung 7 Ausschnitt des Editors für die Regelsets

4.2.3 Auswahl zu platzierendes Objekt

Hier musste die ursprünglich geplante Funktionalität eingeschränkt werden. Da man die Objekte nicht vorhersagen kann, ist es nicht möglich, eine vorgefertigte Liste mit Symbolen und Grössen zu verwenden. Im derzeitigen Rule Composer kann der Nutzer frei entscheiden, wie er sein zu platzierendes Objekt definiert. Je nachdem wie ein Parser später das vom Rule Composer generierte JSON einliest, können trotzdem zusätzliche Eigenschaften definiert werden.

4.2.4 Speichern des Output JSON

Wie vorgesehen ist der Aufwand für diesen Punkt sehr tief.

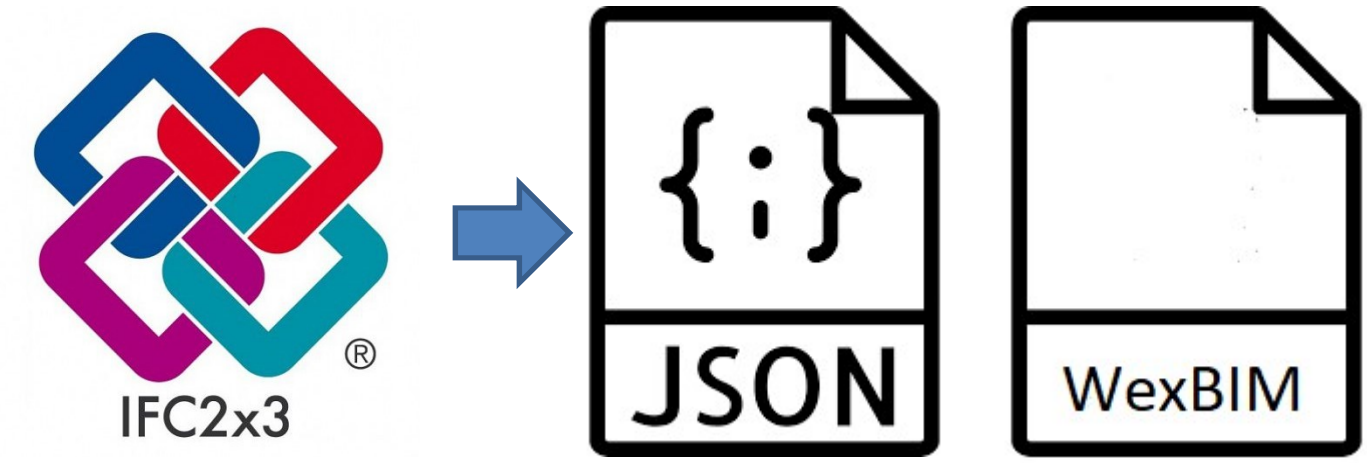

Abbildung 8 Konvertierung von IFC zu JSON und WexBIM

4.2.5 Konvertierer IFC zu WexBIM und JSON

IFC Files sind auf Speicherverbrauch optimiert. Jedoch ist für xBIM WebUI die Performance wichtig, da das 3D-Modell in Echtzeit im Browser gezeichnet werden muss. Deshalb wird für das Zeichnen selber ein „.wexbim“ und für das semantische Modell ein „.json“ erstellt.

4.2.6 Darstellung des 3D Modells

Die Darstellung des BIM Gebäude funktioniert tadellos. Jedoch ist beim Anwählen eines Spaces nicht gut ersichtlich, wo genau und wie gross er ist. Die Idee ist, dass der Space speziell eingefärbt wird oder Wände transparent werden. Dies stellte sich als sehr aufwendig heraus. Im Verlauf von der Arbeit wurde ausschliesslich mit dem semantischen Modell gearbeitet. In dem 3D Modell wurde nie etwas verändert. Deshalb wurde dieses Arbeitspaket nicht bearbeitet.

4.2.7 Sicherstellen der Kompatibilität

Der Output im JSON Format soll auch nach weiterem Bearbeiten unbedingt mit xBM kompatibel bleiben, da man so nach dem Verteilen der Objekte das JSON wieder in den Webviewer zurückladen kann.

4.2.8 Deployment auf aws elastic beanstalk

Das Deployment auf aws (Amazon Web Services) hat der Firma Siemens AG sehr geholfen, ihre Standpunkte überall auf der Welt auf einen gemeinsamen Nenner zu bringen, was die Informatikinfrastruktur betrifft. Deshalb wurde auch der Rule Comopser auf die Cloud geladen. Dies stellte sich dank Amazon elastic beanstalk als unproblematisch heraus. Die asp.net Webanwendung konnte direkt hochgeladen und deplolyed werden. Sie ist natürlich ohne Siemens Zugang nicht erreichbar.

4.2.9 Tests

Zum Testen des Rule Composers kommt nur Integrationstest infrage, weil bei dieser Arbeit sehr viel mit Benutzerinteraktion gearbeitet wird. Das in xBIM vorhandene Unit Test Framework ist mit dem Rule Composer nicht kompatibel. Die ausführlichen Abläufe für die Integrationstests wurden nach Fertigstellung des ersten Prototyps geschrieben

4.3 VERGLEICH XML VORGABE JSON RESULTAT

Es wurde ein von Hand geschriebenes XML File als Beispiel zur Verfügung gestellt. Mit dem Rule Composer wurde versucht, diese Vorlage nachzustellen, um so die Validität zu überprüfen.
Wie man sieht, sind die meisten Parameter abgedeckt. Das JSON hat im Vergleich zum XML deutlich weniger Overhead. Die Parameter haben zusätzlich alle noch Operatoren, welche das JSON noch mächtiger machen. Ursprünglich konnte man mit dem *Tuple* Tag noch mehrere Spaces zusammenhängen. Dies geht leider nicht mehr. Dafür kann man mit dem Unterbaum noch zusätzliche Eigenschaften dazuhängen, wie zum Beispiel Möbel oder Fenster.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Rule name="detector placement_munich">
    <Tuple>
        <ApplicationPoint name="base">
            <Patterns>
                <Pattern type="*">
                    <IFCDISTRIBUTIONELEMENT>
                        <Property type="FDMH291-R"/>
                    </IFCDISTRIBUTIONELEMENT>
                </Pattern>
            </Patterns>
        </ApplicationPoint>
    </Tuple>
    <Actions>
        <Action>
            <FunctionCall name="place_device">
                <Parameters>
                    <Parameter name="device" value="HMS-S">
                    <Parameter name="in_room" value="NULL">
                    <Parameter name="unit" value="m">
                    <Parameter name="at_height" value="0.0">
                    <Parameter name="attach_to" value="base">
                </Parameters>
            </FunctionCall>
        </Action>
    </Actions>
</Rule>
```

Abbildung 9 Beispiel Resultat als XML

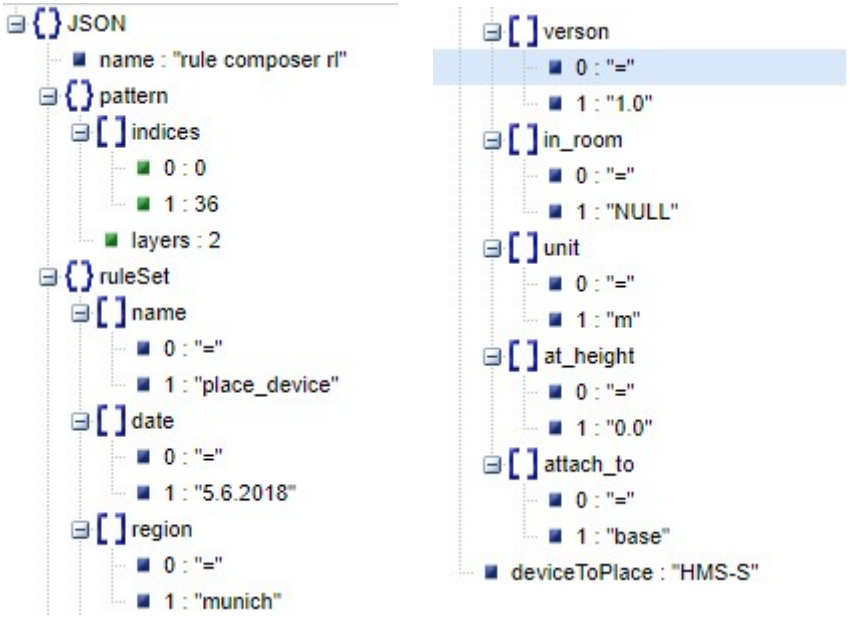

Abbildung 10 Endresultat als JSON

4.4 Schlussfolgerung

Wie man Anhand des Vergleiches des vorgegebenen XML Files und des resultierenden JSON Files erkennt, ist der aus dieser Arbeit hervorkommende Rule Composer äussert flexibel. Dies bedeutet, dass der Generalunternehmer sehr frei ist, was das Gestalten der Regeln angeht. So können selbst zurzeit unvorstellbare Szenarien abgedeckt werden. Der Nachteil ist, dass sich der Generalunternehmer an die Vorschriften des späteren Parsers halten muss, da der Rule Composer die Eingaben nicht stark überprüft.

5. INSTALLATIONSANLEITUNG

5.1 CONVERTIFCTOWEXBIM

Voraussetzung ist Microsoft Visual Studio 2017. Als Version reicht die Community Edition. Ausserdem wird das zu Konvertierende IFC benötigt. Dieses File in den Unterordner *bin/Debug* kopieren. Sobald *Program.cs* ausgeführt worden ist, liegen das „.wexBIM“ und das „.JSON“ im gleichen Ordner bereit.

5.2 XBIMWEBUI

Auch hier ist die Voraussetzung Microsoft Visual Studio 2017. Zusätzlich wird ein aktueller Webbrowser benötigt. Entwickelt wurde auf dem Google Chrome Browser. Die asp.net Anwendung kann aus dem Programm gestartet werden. Der Rule Composer befindet sich unter „http://localhost:xxxx/“. Dort kann man bei den Spaces rechts den Raum aussuchen und daraus eine Regel erstellen. Falls man ein eigenes Model laden möchte, muss man das „.wexBIM“ und das „.json“ File in den Ordner *tests/data* legen. Danach muss man im *browser.js* auf den Zeilen 134 und 135 das eigene File angeben.
xBIM WebUI hat weiter folgende Unterseiten:

- /index.html
- /simpleviewer.html
- /Speedtest.html
- /viewer3d.html

6. INTEGRATIONSTEST ABLÄUFE

Folgende Abläufe wurden getestet:

- User wählt ein Space aus
- Überprüfen von Space Bezeichnungen
- Checkbox Tests der Unterbäume
- Auswahl des Regelsets
- Eingabe des zu platzierende Objektes
- Herunterladen des JSONs
- Überprüfung des Inhaltes von dem JSON
- Editieren von sämtlichen Unterregeln
- Einfügen von zusätzlichen Unterregeln
- Löschen von zusätzlichen Unterregeln
- Erstellen und Löschen von Regelsets
- Speichern der Regelsets
- Abbruch des Editierens von Regelsets

7. PROTOKOLLE DER SITZUNGEN

Auszüge aus den wichtigsten Sitzungen:

- 9. März 2018: Einführung ins Thema, Erstellen des Siemens Batches, Einführung in aws, Einführung in digital-twins, IFC Übersicht
- 27. März 2018: Besprechung mit Herrn Mayer, Überarbeitung PoC Scenario, Device Positioning Process, Einarbeitung in xBIM
- 7. Mai 2018: Zwischenstand des Editors, Präsentation einzelner Komponenten
- 23. Mai 2018: Einführung in die Graphen Datenbank
- 29. Mai 2018: Präsentation ConvertIFCtoWexBIM, Besprechung der Anforderungen an Webserver, Diskussion Designentscheidungen
- 4. Juli 2018: Schlusspräsentation Rule Composer, Abgabe

8. GLOSSAR

- Industrie 4.0: Industrie 4.0 bezieht sich auf die industrielle Fertigung. Der Begriff steht für die vierte industrielle Revolution und wurde erstmals im Jahr 2012 in Deutschland benutzt.
- Internet of Things: Sammelbegriff für kleine Elektronische Geräte, welche in einem Netz verbunden sind, wie zum Beispiel Feuermelder.
- Smart Home: Gebäude mit vernetzter Gebäudetechnik.
- BIM: Building Information Management ist die digitale Revolution der Bauwirtschaft
- Open Source: Open Source Software hat Quelltext, welcher öffentlich und von Dritten eingesehen, geändert und genutzt werden kann
- CAD: Computer-Aided Design ist rechnerunterstütztes Konstruieren
- IFC: IFC Files (Industry Foundation Classes) werden zum Beschreiben von digitalen Gebäudemodellen verwendet.
- XML: Die „Extensible Markup Language“ ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten.
- JSON: Die „JavaScript Object Notation“, ist ähnlich wie XML eine Auszeichnungssprache und gilt als sein Nachfolger
- TypeScript: TypeScript ist eine typisierte Programmiersprache, welche sich zu JavaScript kompilieren lässt und im Browser ausführbar ist.
- Generalunternehmer: Ein Generalunternehmer ist der Leiter eines Baus, er ist auch für die Gebäudetechnik zuständig, deshalb werden sie die Hauptnutzer des Rule Composers sein.
- DOM: Das „Document Object Model“ ist die Schnittstelle zwischen HTML und JavaScript
- AWS: „Amazon Web Services“ ist die Cloud von Amazon.

- Elastic Beanstalk: Elastic Beanstalk ist ein Service zum Bereitstellen und Skalieren von Webanwendungen in vielen Sprachen.

9. QUELLEN

- https://www.swissmem.ch/de/industrie-politik/industrie-40-digitalisierung.html
- http://docs.xbim.net/
- https://github.com/xBimTeam/XbimWebUI
- https://en.wikipedia.org/wiki/Industry_Foundation_Classes
- https://en.wikipedia.org/wiki/Building_information_modeling
- http://docs.xbim.net/XbimWebUI/
- https://www.bentley.com/en/products/product-line/utilities-and-communications-networks-software/bentley-openutilities-designer
- https://www.solibri.com/de/
- https://www.siemens.com/customer-magazine/en/home/industry/digitalization-in-machine-building/the-digital-twin.html

10. ANHANG

10.1 PERSÖNLICHE BERICHTE

Als ich mit dem Projekt begann, war mir relativ schnell klar, was gemacht werden muss. Nur die Technologie war nicht klar. Nach anfänglichem Einlesen und Einarbeiten mit dem Bentley Tool welches bisher für die Regelerstellung gedacht war, bekam ich die Mitteilung, dass ich mit xBIM arbeiten soll. Da xBIM Open Source ist, war es jedoch einfach für mich einen Einstieg zu finden. Während des Verlaufs der Arbeit wurde mir immer bewusster, dass das Essenziellste der Arbeit das Finden der Patterns ist. Die Regelsets sind auch sehr wichtig und aufwändig. Die Regelsets sind relativ offen und der spätere Parser muss sich um diese kümmern. Bei der Schlussbesprechung stellte sich heraus, dass der Rule Composer das ist, was sich die Firma Siemens vorgestellt hatte.

10.2 EIGENSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum:
Wilen, 9.7.2918
Name, Unterschrift:
Roman Lacher

10.3 AUSZUG AUS DER SIEMENS GUI GUIDELINE

Folgen auf den Nächsten Seiten

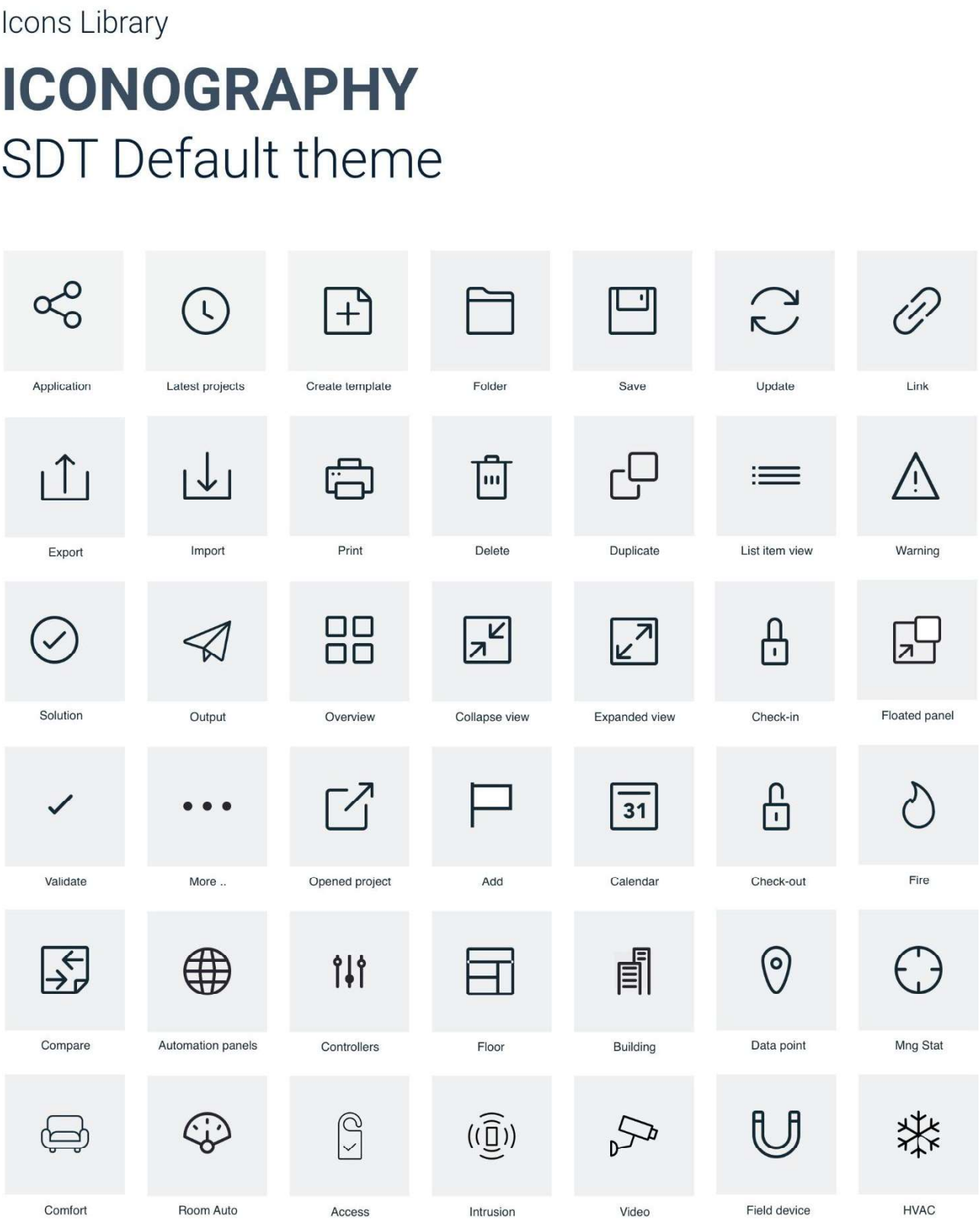

Figure 64: UI Icons library.

5.1.5.1 Style (Measurement and color)

These are the styles and measurements for the basic states that the UI icons can have: default, hover and disabled. Any additional state that may be needed by particular components will be described in their corresponding sections.

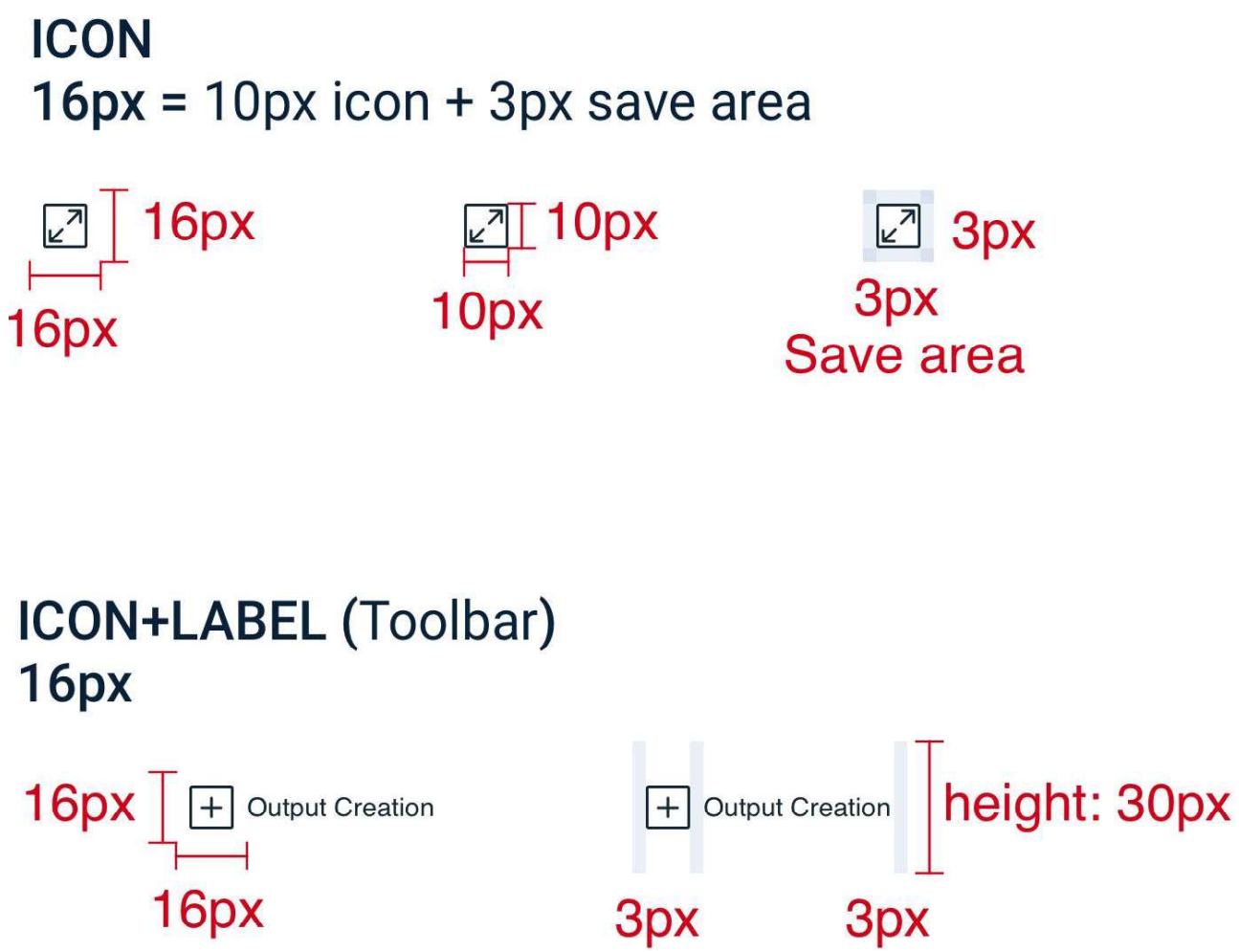

Figure 65: UI Icons measurement.

Color and Typography

Attribute	Color	Typography (font size px/pt.) / Measurement
Normal state	R: 90 G: 104 B: 114	Label size. 6px
Hover state	R: 21 G: 41 B: 52	Label size. 6px
Disable state	R: 140 G: 155 B: 165	Label size. 6px

Standard icons will be used for application global actions, saving a project, exporting or creating a project, and for filtering content by discipline. Utilities and action icons will be used in concrete contexts to manage lists of elements.

Structure

Attribute	Height / Width	Spacing
Icon	16px (10px icon + reserved area)	Margin-bottom: 3px (between reserved space (3px) for the icon and text label (1px))
Icoin +label	16px (10px icon + reserved area)	Margin-left: 3px (between reserved space and text label (3px))

[Icon Library will be feed and complemented continuously during the next steps.]

5.1.6 Typography

Typography is used to create clear hierarchies, useful organizations, and purposeful alignments that guide users through the product and experience. It is the core structure of any well-designed interface.

System Design Tool will use **Helvetica** font-face family available in Perspectix font Catalog.

Helvetica font family has the objective and functional style which was associated with Swiss typography in the 1950's and 1960's. It has been adapted over the years for all methods of composition.

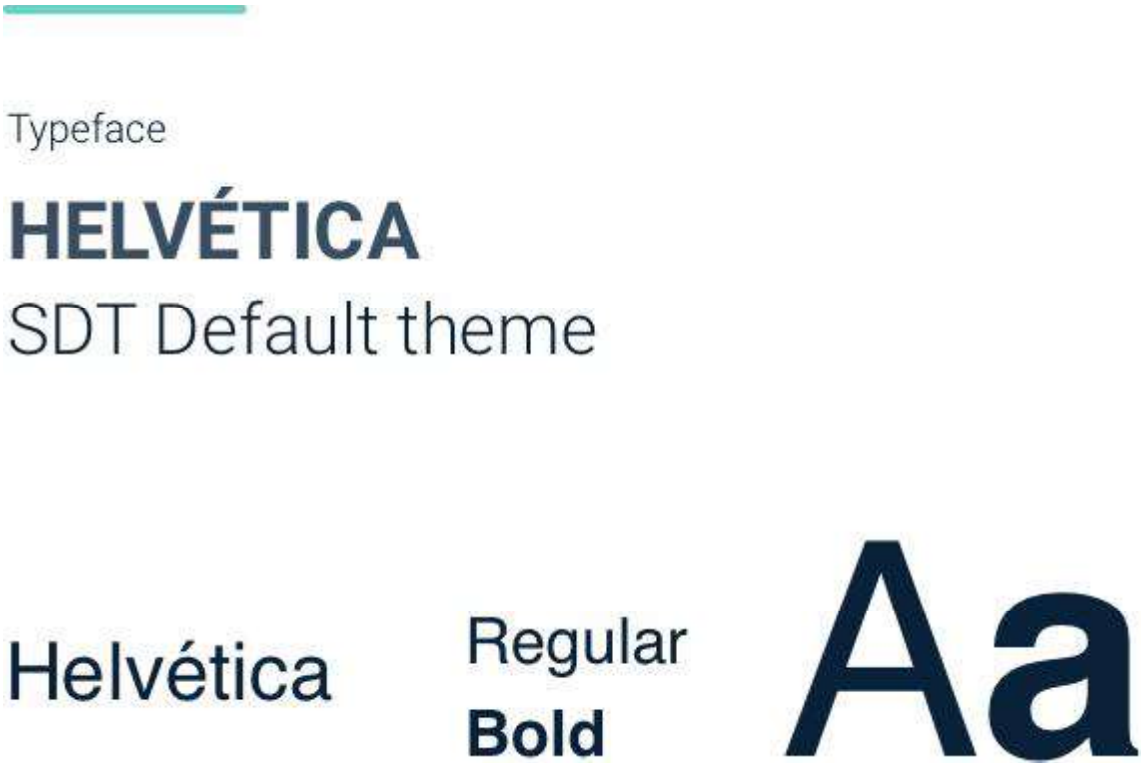

Figure 66: Helvetica Font available in Perspectix font-face Catalog.

Font size is specified in units called points. Points size refers to the size of the font, which is roughly 1/72 inch.

5.1.6.1 Usage patterns and guidelines

Font size measurements had been considered in pixels in this document to display an exact font size measure and it does not depend on the screen resolution.

Typography has several use patterns and guideline:

Attribute	Color (RGB)	Font Size* and Style

Attribute	Color (RGB)	Font Size* and Style
Body Icon label Label	RGBA: #152934FF	Helvetica (8px)
Nav-menu	R: 90 / G: 104 / B: 114	Helvetica (8px)
CATEGORY	RGBA: #8C9BA5FF	Helvetica (6px)
Tabs	RGBA: #152934FF	Helvetica bold (10px)
TITLE PANEL	RGBA: #152934FF	Helvetica (8px) –Uppercase – Letter-spacing: 0.8
Tree Label – Level1	RGBA: #152934FF	Helvetica (8px)
Tree Label – Level1 :selected	RGBA: #152934FF	Helvetica bold (8px)
Tree Label – Level 2	RGBA: #152934FF	Helvetica (8px) **Example style for all descendent nodes.*
Links	RGBA: #5596E6FF	Helvetica (8px)
Links (:hover)	RGBA: #0A2239FF	Helvetica (8px)

**Point Measurement is relative to screen resolution*.

5.2 Accessibility

System Design Tool is committed to following and complying with the best practices when it comes to accessibility. Each component and element were built following the Web Content Accessibility Guidelines (WCAG) and met the AA standard.

Our patterns are perceivable, operable, and understandable to users, providing support for keyboard interactions and screen readers.

5.2.1 Keyboard

Enabling standard keyboard access methods, the user should be able to transverse the entire set of user interface controls and content areas using keyboards keys as:

Keyboard key	Description
Ctrl key	Copy an element
Shift key	Multi-drop the selected elements
Ctrl + Delete key	Delete selected element
Ctrl + D key	Duplicate elements
Cursor key	Up and Down key to move in the listed items once there is at least one element selection
Ctrl + N	Create a new project from scratch
Ctrl + O	Open a project, displays an open dialog.

Keyboard key	Description
Ctrl + S	Save the project
Ctrl + Q	Quit application
Ctrl + Z	Undo user actions

5.3 Components

This section points out detailed setting of components or layout elements that are displayed in System Design Tool, making a comparison between the proposal and the current view in PX'5, dealing with measurements and style of the component or layout.

5.3.1 Accordion

An accordion allows a user to toggle the display of a content section.

Emergya UI Proposal

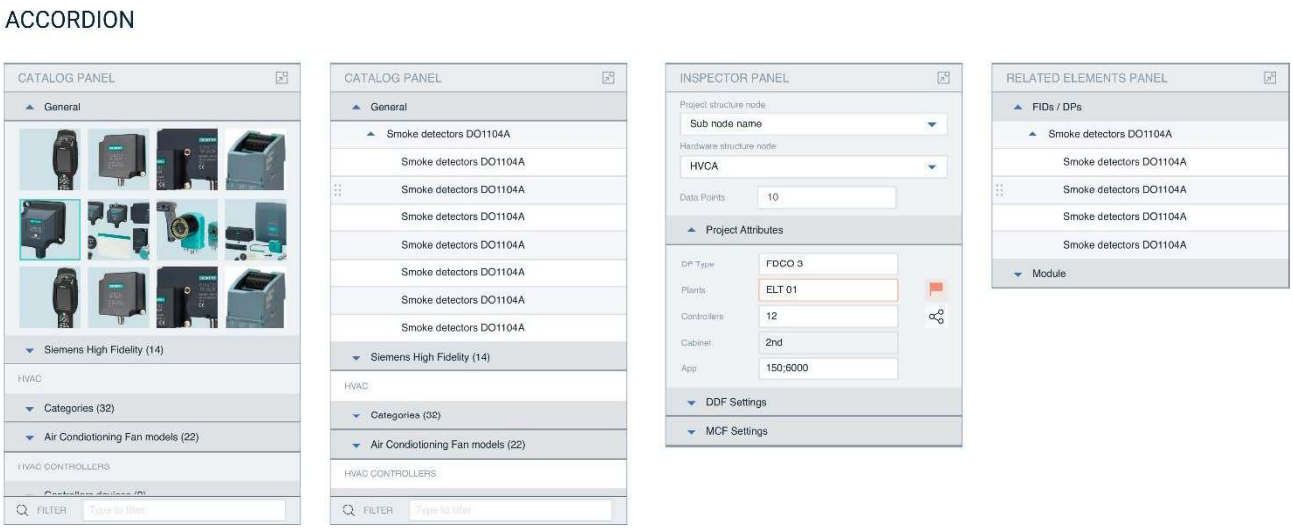

Figure 67: Accordion proposal

5.3.1.1 Style (Measurement and color)

Accordions are used to group specific content within a page or when vertical space is limited.

The user may have multiples expanded Accordions at the same time.

The entire Title area is clickable area to expand or collapse the content below.

Color and Typography

Attribute	Color	Typography (font size px) / Measurement
Label	RGBA: #152934FF	Helvetica regular (7px), lines height (8px) left alignment.
Body	RGBA: #152934FF	Helvetica regular (7px), lines height (8px) left alignment.
CATEGORY	RGBA: #8C9BA5FF	Helvetica regular (6px), lines height (10px) left alignment.
Separator lines	RGBA: #8C9BA5FF	Border: 1px

Attribute	Color	Typography (font size px) / Measurement
Group or Category Background	RGBA: #DFE3E6FF	--
Background Zebra style (items)	RGBA: #FFFFFF RGBA: #F5F7FAFF	--
Separator lines	RGBA: #DFE3E6FF	Border: 1px

Structure

Attribute	Height	Spacing
Title area	20px	5px (top and bottom spacing)
Expand icon – Label	--	3px
Content	--	10px (left and right spacing) 5px (top and bottom spacing)

ACCORDION

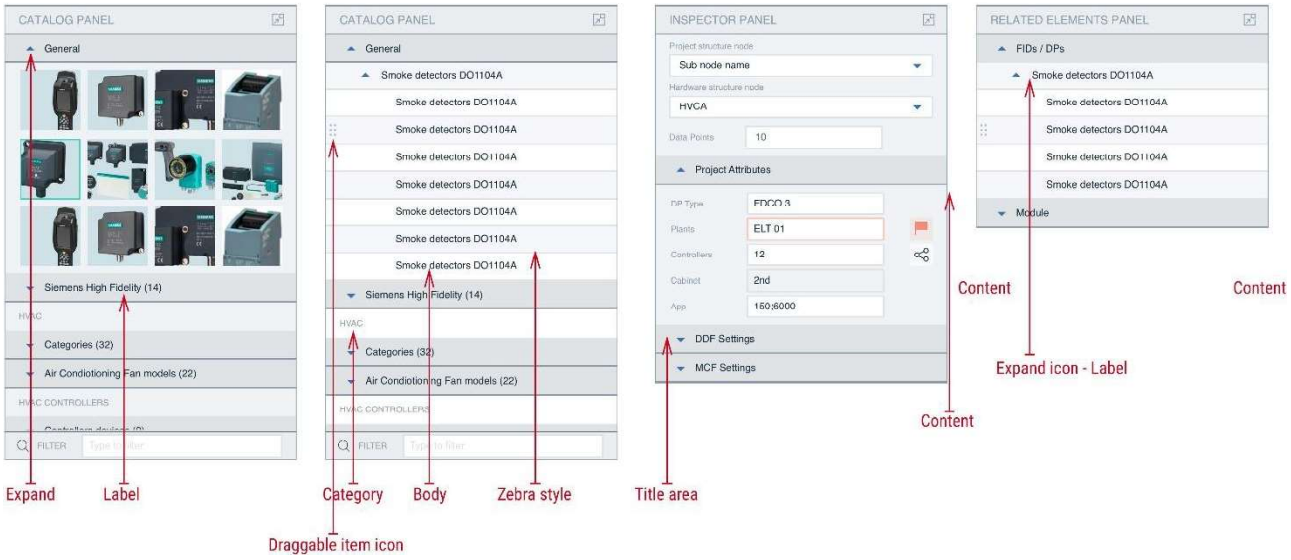

Figure 68: Accordion Style proposal

5.3.2 Button

Buttons are used to invoke an event or action.

Emergya UI Proposal

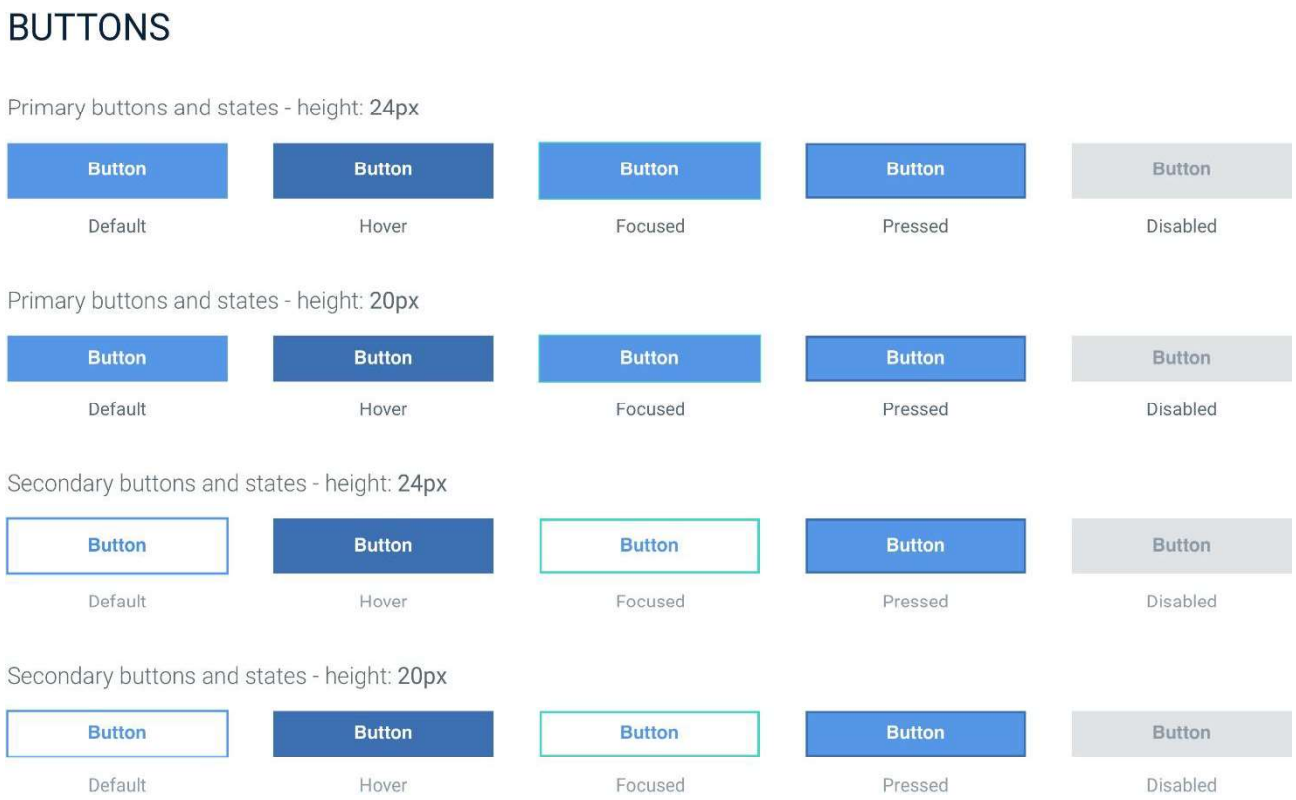

Figure 69: Buttons proposal.

5.3.2.1 Style (Measurement and color)

Button width will depend on the label length.

Primary buttons - Color and Typography

Attribute	Color	Typography (font size px) / Measurement
Text Normal / Hover	RGBA: #FFFFFF	Helvetica bold (7px), center alignment.
Text Disable	RGBA: #8C9BA5FF	Helvetica bold (7px), center alignment.
Background Normal State	RGBA: #5596E6FF	--
Background Hover State	RGBA: #0A2239FF	--
Background Focused State	RGBA: #5596E6FF Focused: RGBA: #D6FFFAFF (border)	Border: 1px
Background Pressed State Border	RGBA: #5596E6FF Border: RGBA: #0A2239FF	Border:1px
Background Disable State	RGBA: #DFE3E6FF	--

Secondary buttons - Color and Typography

Attribute	Color	Typography (font size px) / Measurement
Text Normal	RGBA: #5596E6FF	Helvetica bold (8px), center alignment.
Text Hover	RGBA: #FFFFFF	Helvetica bold (8px), center alignment.
Text Disable	RGBA: #8C9BA5FF	Helvetica bold (8px), center alignment.
Border line Normal	RGBA: #5596E6FF	Border style 1px solid
	RGBA: #5596E6FF Focused: RGBA: #D6FFFAFF (border)	Border: 1px
Background Focused state	RGBA: #FFFFFF	--
Pressed State (Background and Border	RGBA: #5596E6FF Border: RGBA: #0A2239FF	--

Structure

Attribute	Height	Spacing
Button height	24px	8px (left, top, right and bottom spacing)
Button height (small)	20px	6px (left, top, right and bottom spacing)

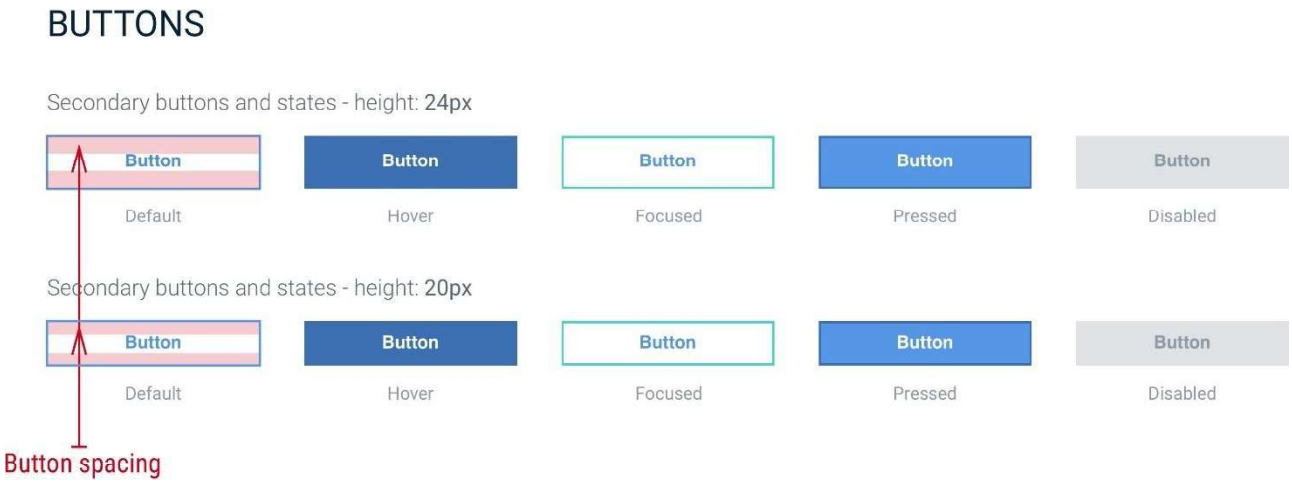

Figure 70: Button style proposal.

5.3.3 Checkbox

Checkboxes are used when there is a list of options and the user may select multiples options, including all or none.

Emergya UI Proposal

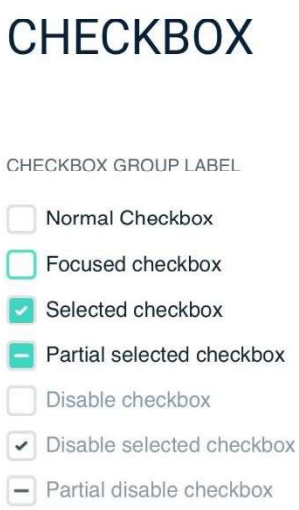

Figure 71: Checkbox proposal

5.3.3.1 Style (Measurement and color)

A heading can accompany a set of Checkboxes to provide further context or clarity. Use a sentence case for Checkbox heading.

Checkbox label must be explicit and clear about the action that will follow.

Color and Typography

Attribute	Color	Typography (font size px) / Measurement
Label	RGBA: #152934FF	Helvetica regular (7px), lines height (8px) left alignment.
Disabled Label	RGBA: #8C9BA5FF	--
Marked Checkbox fill	RGBA: #D6FFFAFF	--
Focused Checkbox	RGBA: #D6FFFAFF	--
Check or partial icon selected	RGBA: #FFFFFF	--
Checkbox fill	RGBA: #FFFFFF	--
Check or partial icon – disable	RGBA: #152934FF	--
Checkbox border	RGBA: #DFE3E6FF	1px

Structure

Attribute	Height	Spacing
CheckBox	10px	5px (left, top, right and bottom spacing)
Box to Label		3 px

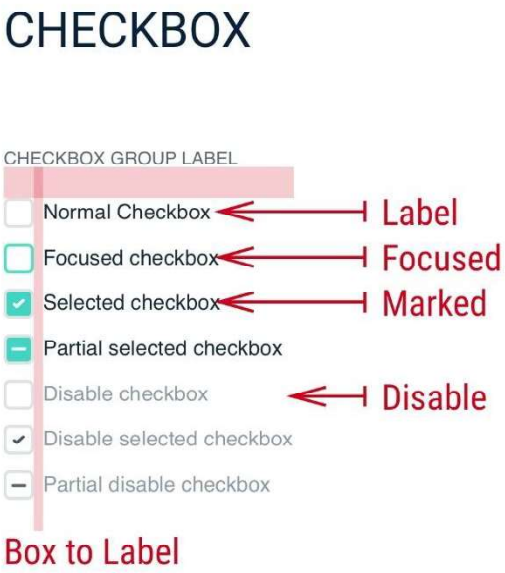

Figure 72: Checkbox style proposal

5.3.4 Date picker

A date picker is a text input to capture a date. You can select a single date or date range.

Emergya UI Proposal

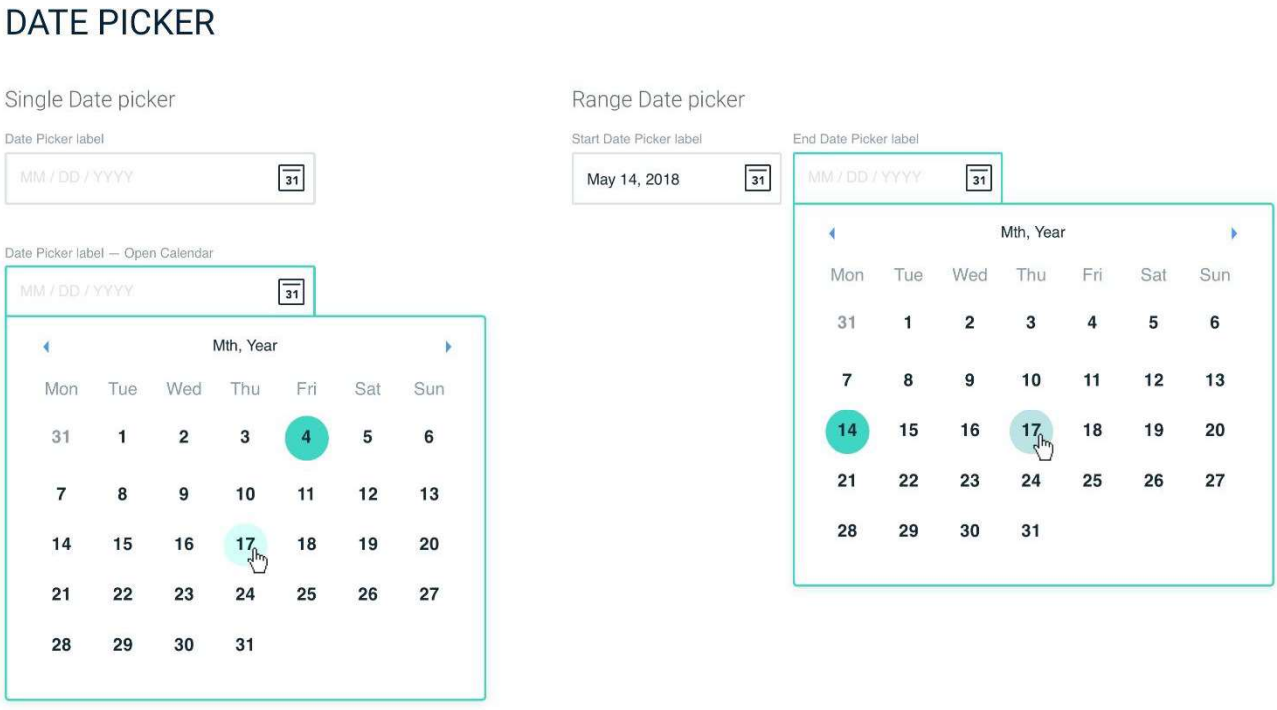

Figure 73: Single and Range Date picker – Emergya proposal.

5.3.4.1 Style (Measurement and color)

Date picker uses a placeholder text to explain the user how to input in the correct format.

Color and Typography

Attribute	Color	Typography (font size px) / Measurement
Label	RGBA: #8C9BA5FF	Helvetica (6px), left alignment. Line-height: 8px
Placeholder	RGBA: #DFE3E6FF	Helvetica (6px), left alignment. Line-height: 8px
Date picker text (month/year and day)	RGBA: #152934FF	Helvetica regular (7px), lines height (8px) left alignment.
Date picker text: day of the week – regular state	RGBA: #152934FF	Helvetica bold (7px), lines height (8px) left alignment
Date picker text: day of the week – hover state	RGBA: #D6FFFAFF	Helvetica bold (7px), lines height (8px) left alignment
Background selected state	RGBA: #C1EDE8FF	20px
Background hover state	RGBA: #D6FFFAFF	20px
Border color – expanded Data picker	RGBA: #C1EDE8FF	1px

Structure

Attribute	Height	Spacing
Label and input box	--	3px (between label and input box)
Icon	16px	Left-padding: 10px (with the input-box) Right-padding: 3px
Single Date picker	Input: 20px Selected area: 20px	Row spacing: 10px Column spacing: 8px
Range Date picker: Label and input-box	--	Margin-bottom: 5px
Range Date area	20px	Margin-left: 5px