Politechnika Poznańska

Dokumentacja

Tablica informacyjna

Autor:
Damian Antczak
Szymon Kłębowski

September 30, 2016

Contents

1 Wprowadzenie

Projekt ma na celu zamianę telewizora lub monitora w tablicę informacyjną wykorzystywaną np. w poczekalnie, na uczelni, w szkole itd. Funkcjonalność tą można uzyskać poprzez podłączenie ekranu do Raspberry Pi lub innego jedno-płytkowego komputera, z systemem operacyjnym Linux. Zarządzanie aplikacją odbywa się poprzez serwis www, w którym to definiujemy, jakie moduły mają być wyświetlane na ekranie. Użytkownik ma do dyspozycji moduły wyświetlające zdjęcia, RSS, dane pogodowe oraz jednostronicowe pliki HTML. Zapisywany jest również zrzut ekranu, które następnie wyświetlany jest w panelu administratora.

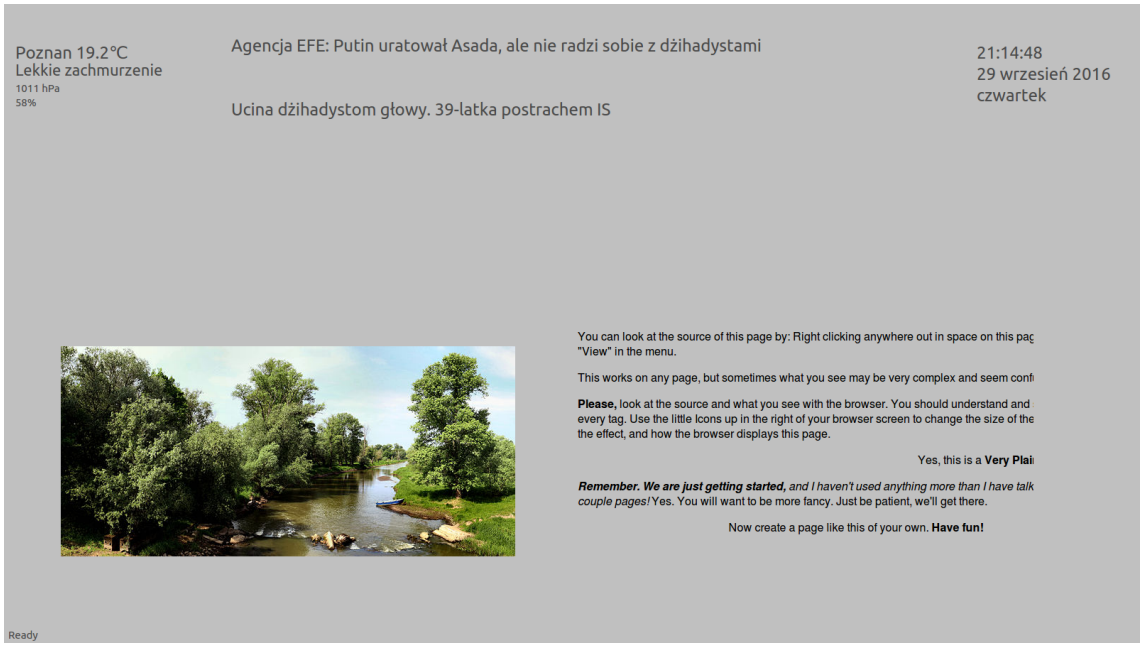

Figure 1: Wygląd ekranu

2 Impementajca

2.1 Część embedded

Cześć odpowiedzialna za wyświetlanie informacji na ekranie została napisana w języku Python używając biblioteki Qt. Jest to wielkoformatowa biblioteka do tworzenie interfejsu graficznego. Nakładka na bibliotekę Qt dla języka python nazywa się PyQt. Biblioteka PyQt zapewniała łatwość implementacji dzięki wbudowanej obsłudze multimediów, timerów, czy zintegrowanemu silnikowi WebKit do wyświetlania HTML.. W projekcie zostały użyte również inne biblioteki Pythona: urllib do obsługi zapytań HTTP, json do parsowania pilków json, feedparser do parsowanie RSS.

2.1.1 Najważniejsze fukcję programu

Wszystkie elementy wyświetlane na ekranie są instancjami klas dziedziczących po klasie QWidget. Jest to klasa bazowa wszystkich obiektów interfejsu użytkownika. Dzięki zastosowaniu takiego rozwiązania została osiągnięta wysoka przejrzystość kodu oraz możliwość łatwego rozszerzenia go o kolejne widgety.

```
DateTimeWidget(self,parent=None, x=0, y=0)
```

- widget który wyświetla aktualna date i czas

```
RssWidget(self,parent=None,x=0,y=0,url="URL")
```

-widget, który wyświetla informację z kanału RSS, do parametru url podajemy URL RSS.

```
WeathercastWidget(self,parent=None,x=0,y=0,url="URL")
```

-widget wyświetlający pogodę z openweathermap zmieniając URL można wyśietlać pogodę np. z różnych miast

```
ImageWidget(self,x,y)
```

-widget umożliwa wyświetlanie zdjęć znajdujących się w katalogu images. Zdjęcia zmieniają się automatycznie po czasie 2 sekund.

```
HtmlWidget(self,parent=None, x=0, y=0, url="URL")
```

-widget wyświetlający proste pliki HTML

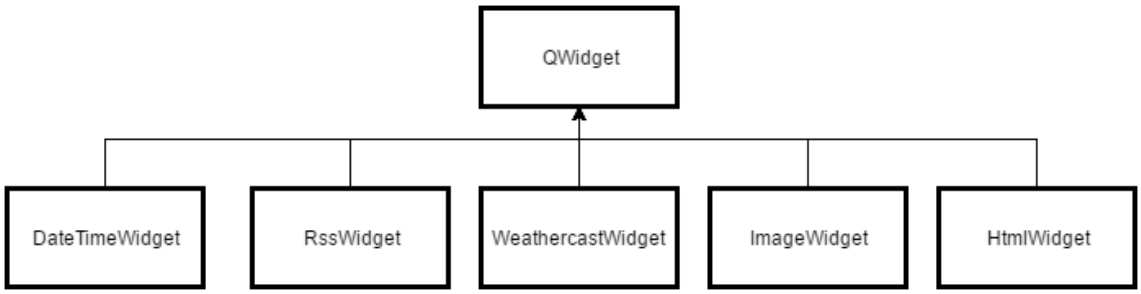

Figure 2: Klasy widget

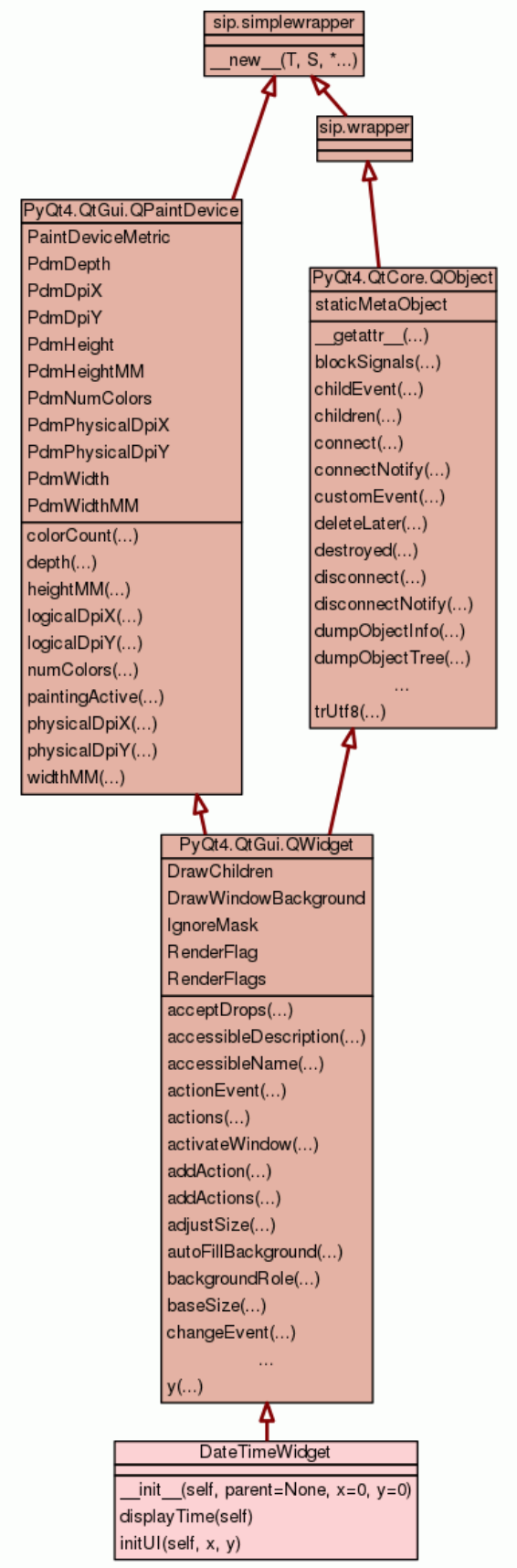

Figure 3: DateTimeWidget

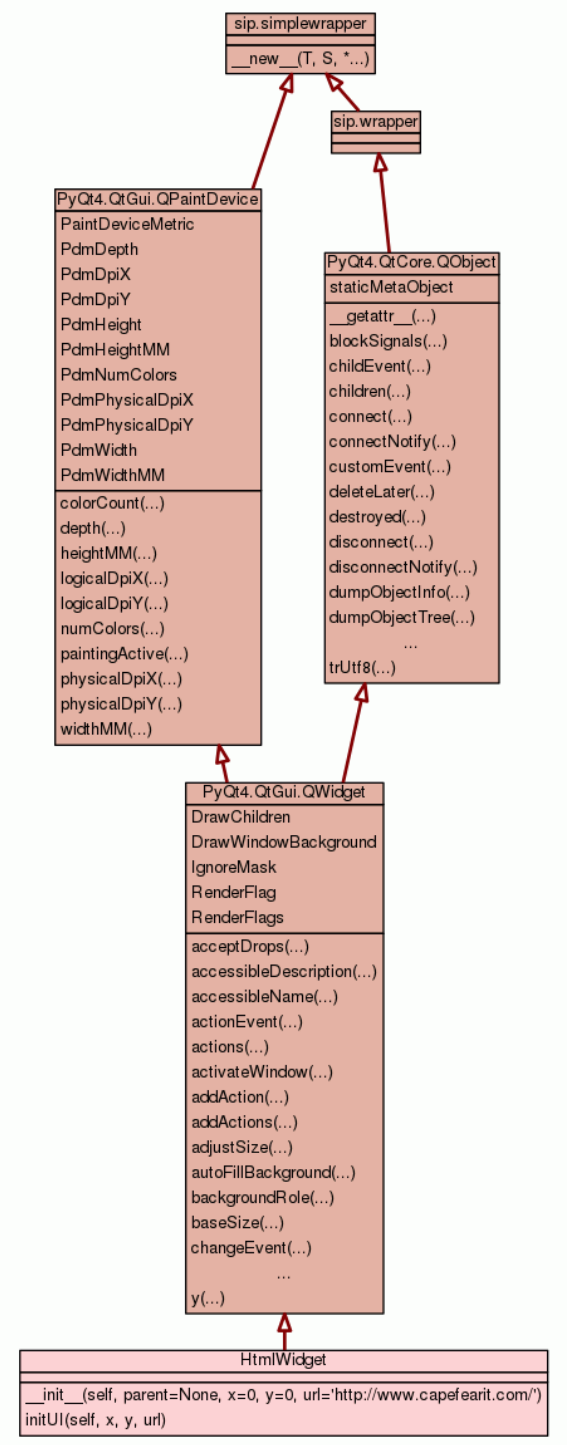

Figure 4: HtmlWidget

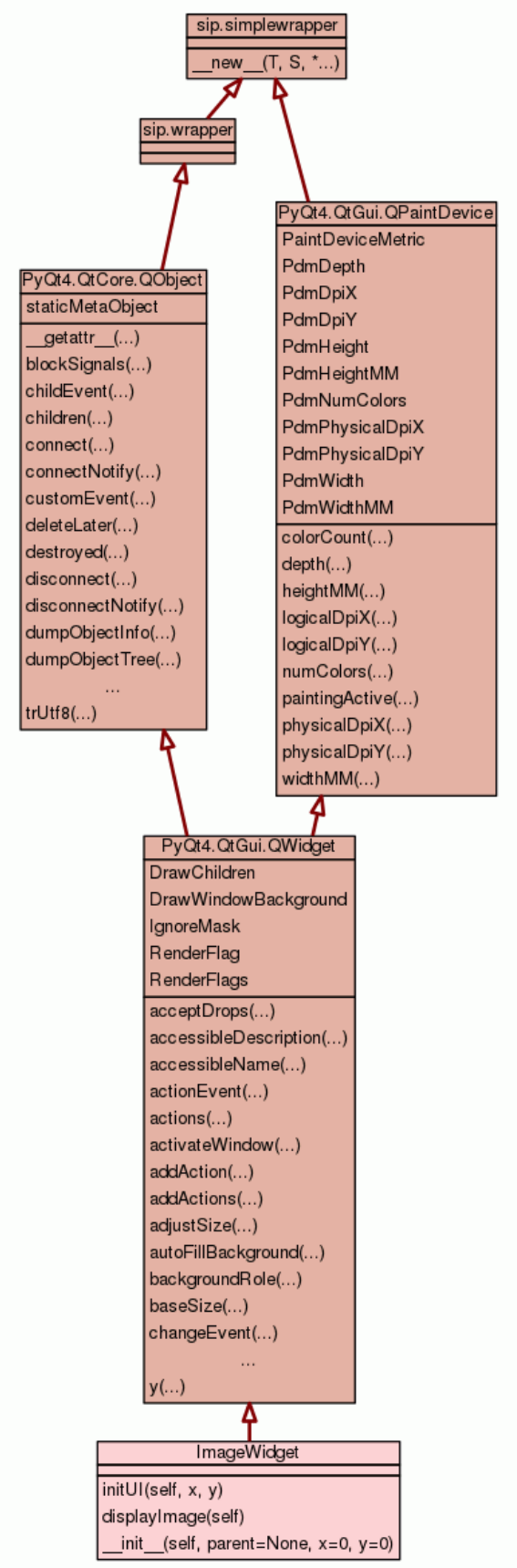

Figure 5: ImageWidget

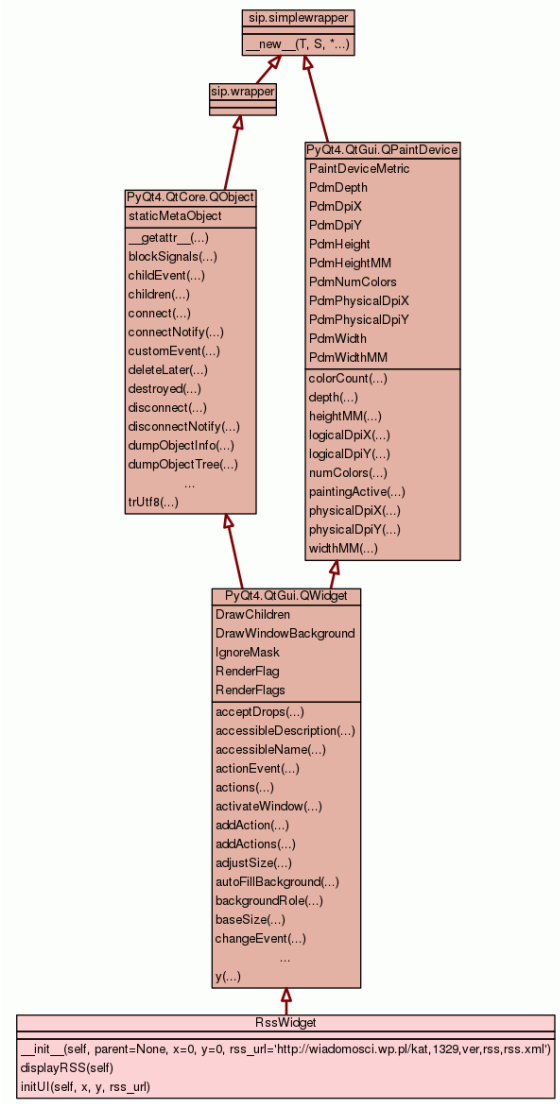

Figure 6: RssWidget

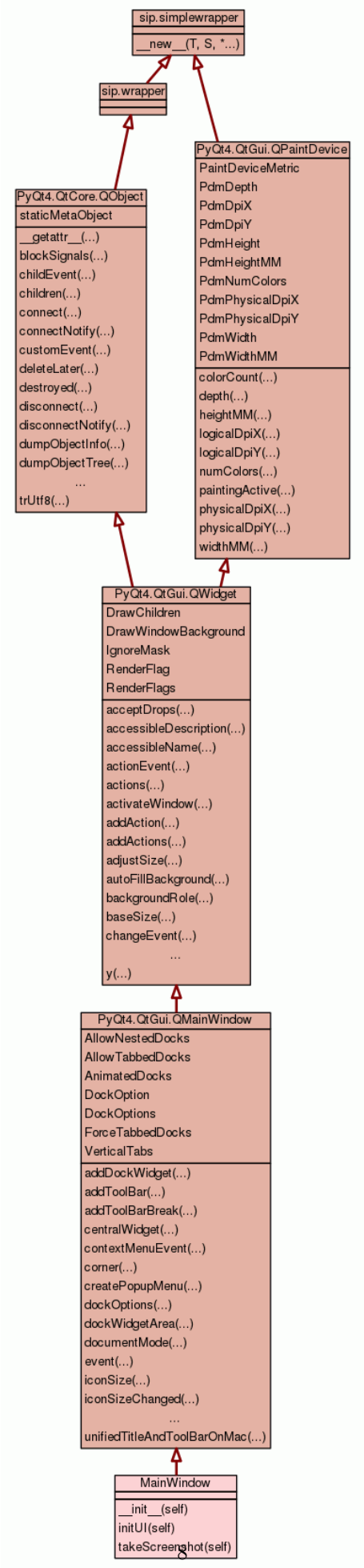

Figure 7: MainWindow

2.1.2 Definiowanie interfejsu za pomocą XML

Elementy wyświetlane na ekranie definiowane są w pliku XML o poniższej strukturze.

```
<table>
    <page>
        <style>background-color: silver</style>
        <widget name="WeathercastWidget" x="1%" y="5%" ></widget>
        <widget name="RssWidget" x="20%" y="5%" ></widget>
        <widget name="RssWidget" x="20%" y="15%" extra="'http://fakty.interia.pl/
    feed'"></widget>
        <widget name="ImageWidget" x="5%" y="50%"></widget>
        <widget name="DateTimeWidget" x="90%" y="5%"></widget>
        <widget name="HtmlWidget" x="50%" y="50%" url="'http://www.zyvra.org/html/
    simple.htm'"></widget>
    </page>
</table>
```

Do parsowania pliku XML z opisem interfejsu służy klas XMLparser . Instancja tej klasy przegląda drzewo DOM w celu znalezienia znacznika „widget” w którym to zdefiniowane są parametry widgetu. W celu poprawnego zdefiniowana widgetu w znaczniku musi być przynajmniej uzupełniony atrybut „name”, który to określa jaki typ widgetu ma być wyświetlany. Przykładowa definicja widgetu „<widget name="HtmlWidget" x="50%" y="50%" url="'http://www.zyvra.org/html/simple.htm'"></widget>”. Za pomocą atrybutów x i y można ustalać pozycję widgetu na ekranie. Pozycję na ekranie można ustawiać podając pozycję w pixelach np. x=”25px” lub podając pozycję w procentach ekranu np.=”50%”

2.2 Panel administracyjny

Panel administracyjny jest stroną internetową umożliwiającą ustalenie wyglądu tablicy. Do stworzenia panelu użyto HTML, JavaScript oraz PHP.

2.2.1 Wygląd oraz zastowsowanie panelu

Użytkownik ma do wyboru kilka elementów, odpowiadającym widgetą z tablicy, które może dowolnie ustawić w zakresie "tablicy". Przypisanie adresu dla danego widgetu następuje po naciśnięciu przycisku "Zatwierdź adres".
Po naciśnięciu przycisku "Generuj" następuje wygenerowanie kodu XML, który następnie przesyłany jest do skryptu PHP działającego na serwerze w celu zapisu na dysku.

Figure 8: Wygląd panelu administratora

Dla powyższego przykładu zosanie wygenerowany następujący plik XML.

```
<?xml version="1.0" encoding="utf-8" ?>
<table>
<page>
<widget name="HTTPWidget" x="0" y="0" url="wp.pl"></widget>
<widget name="DateTimeWidget" x="529" y="9"></widget>
<widget name="RssWidget" x="467" y="238" url="http://wiadomosci.wp.pl/kat,1329,ver,
    rss,rss.xml"></widget>
</page>
</table>
```

Listing 1: Wygenerowany XML

2.3 Komunikacja

Użytkownik kożysta z serwisu udostępnionego przez serwer oraz pośrednio może zapisać plik konfiguracyjny na serwerze. Użądzenie pobiera z serwera plik konfiguracyjny. Fig 9.

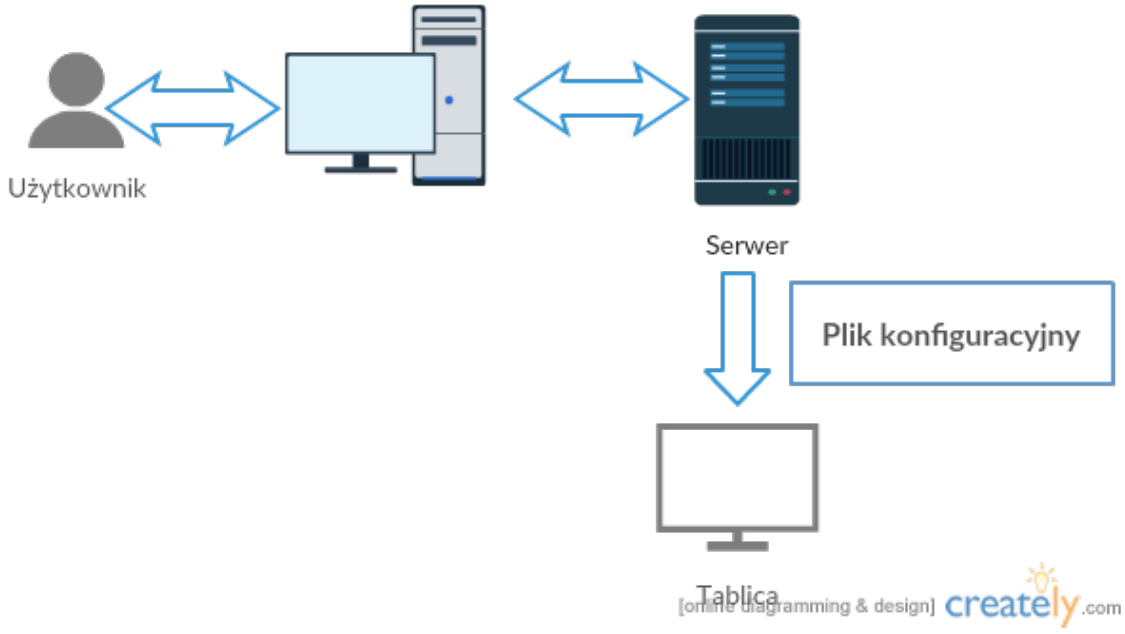

Figure 9: Komunikacja pomiędzy modułami.

2.4 Najważniejsze metody programu

2.4.1 Zarządzanie obietkami na tablicy

Funkcja przedstawiona na listingu 2 definiuje zachowanie obiektów na panelu administracyjny. Dla każdego z elementów listy w odpowiedzi na funkcję "drop" tworzony jest nowy obiekt, który potem jest przechowywany w tabeli, z właściwością "draggable".
abstractname Każdy z widgetów jest rozszerzany o dodatkowe elementy zależne od typu widgetu(np. przycisk zamknięcia), oraz dodatkowe funkcje obsługujące te elementy (delete, confirm). Podczas trwania zdarzenia "drag" odczytywana jest pozycja elementu, a zdarzenie "stop" zapisuje pozycje w tabeli obiektów.

```
                $(".obiekt").draggable({
                    create: function (event, ui) {
                        elem = this;
                        $(elem).position({
                            my: "left top",
                            at: "left top",
                            of: $startPostion
                        });
                        obiect.setClass(checkClass($(elem)));
                        addElementsToDivs($(elem));
```

```
                    widget.push(obiect);
                    $(elem).append('<input type="button" class="delete" value="
X"/>');
                    jQuery(elem).attr("id", id);
                    var localID = $(elem).attr("id");
                    $(elem).children(".delete").position({
                        my: "right top",
                        at: "right top",
                        of: elem
                    });
                    $(elem).children(".delete").click(function () {
                        // xmlTab.splice($localIndex, 1);
                        for (var i = 0; i < widget.length; i++) {
                            if (widget[i].id == localID) {
                                widget.splice(i, 1);
                                break;
                            }
                        }
                        $(elem).closest(".obiekt").remove();
                    });
                    $(elem).children(".confirm").click(function () {
                        for (var i = 0; i < widget.length; i++) {
                            if (widget[i].id == localID) {
                                widget[i].setUrl($(elem).children(".input").val
());

                                break;
                            }
                        }
                    })
                },
                start: function () {
                    $element = this;

                },
                drag: function (event, ui) {
                    pos = $(this).offset();
                    left = (pos.left - $startPostion.offset().left);
                    $top = (pos.top - $startPostion.offset().top);
                    $(".xPos", $element).val(left);
                    $(".yPos", $element).val($top);
                    $(".xD").val("left: " + left.toFixed(0) + " top: " + $top.
toFixed(0));
                },
                containment: "#table",
                stop: function (event, ui) {
                    var x = left.toFixed(0);
                    var y = $top.toFixed(0);
                    for (var i = 0; i < widget.length; i++) {
                        if (widget[i].id == $(this).attr("id")) {
                            widget[i].setPosition(x, y);
                            break;
                        }
                    }
                }

            });
```

Listing 2: Zarządzanie obiektami na tablicy

2.4.2 Generowanie XML

GenerateXML() - listing 3 funkcja generuje kod XML w oparciu o widgety znajdujące się w tabeli. Dla każdego obiektu sprawdzane są wartości „x”, „y” oraz „url”, a kod XML jest rozszerzany w zależności od klasy obiektu.

```
function generateXML() {
    $(area).val("");
    var val = '<?xml version="1.0" encoding="utf-8" ?>\n<table>\n<page>\n';
    var $class = "";
    var x = "";
    var y = "";
```

```
        var url = "";
        for (var i = 0; i < widget.length; i++) {
            $class = widget[i].getClass();
            x = widget[i].getX();
            y = widget[i].getY();
            url = widget[i].getUrl();
            if ($class == 'watch') {
                val = val + '<widget name="DateTimeWidget" x="' + x + '" y="' + y +
     '"></widget>';
            }
            else if ($class == 'rss') {
                val = val + '<widget name="RssWidget" x="' + x + '" y="' + y + '"
    url="' + url + '"></widget>';
            }
            else if ($class == 'weather') {
                val = val + '<widget name="WeathercastWidget" x="' + x + '" y="' +
    y + '" url="' + url + '"></widget>';
            }
            else if ($class == 'pictures') {
                val = val + '<widget name="ImageWidget" x="' + x + '" y="' + y + '"
     url="' + url + '"></widget>';
            }
            else if ($class == 'http') {
                val = val + '<widget name="HtmlWidget" x="' + x + '" y="' + y + '"
    url="' + url + '"></widget>';
            }
            val = val + "\n";
        }
        return val+"</page>\n</table>";
    }
```

Listing 3: Generowanie XML

2.4.3 Zapis pliku konfiguracyjnego

Zapis do pliku na serwerze składa się z dwóch części. Po stronie użytkownika wykonywany jest kod przedstawiony na listingu 4, który wykonuje operację POST do skryptu PHP znajdującego się na serwerze. Skrypt na serwerze (listing 4) odbiera dane i zapisuje je do pliku tablica.txt, sąd można go później pobrać.

```
    function toPHP(xml) {
        var data = new FormData();
        data.append("data", xml);
        var xhr = (window.XMLHttpRequest) ? new XMLHttpRequest() : new
    activeXObject("Microsoft.XMLHTTP");
        xhr.open('post', 'adres_do_skryptu_php', true);
        xhr.send(data);
    }
```

Listing 4: Skrypt publikacji wygenerowanego XML.

```
<?
if(!empty($_POST['data'])){
$data = $_POST['data'];
echo $data;
$path = $_SERVER['DOCUMENT_ROOT'].'/upload/tablica.xml';
echo $path;
$file = fopen($path, 'w+');
if($file === false){
  exit;
}
fwrite($file, $data);
fclose($file);
}
?>
```

Listing 5: Kod PHP zapisujący XML do pliku

2.4.4 Adresy

- Panel administratora

- Plik konfiguracyjny

3 Literatura

- Instrukcja instalacji serwera Nginx oraz php5.
- Dokumentacja jQuery
- Poradnik HTML
- Poradnik JavaScript
- Dokumentacja PyQT
- Stack Overflow