Wstęp

Jak wiadomo, wiele rzeczy w dzisiejszych czasach zostało bardzo zautomatyzowane. Dotyczy to nie tylko miejsc pracy gdzie potrzebna jest bardzo duża precyzja produkcji lub czas pracy, ale także do rutynowych, prostych, aczkolwiek żmudnych zadań. Podobne sytuacje mają także miejsce w systemach komputerowych, w których zachodzi potrzeba regularnego wykonywania czynności takich jak np. tworzenie kopii zapasowych, testowanie funkcjonalności oprogramowania czy weryfikowanie internetowych zasobów. Do takich właśnie celów służą programy automatyzujące, które pozwalają odciążyć użytkownika i ułatwiają mu skupienie się na innych zadaniach. Jednak trzeba pamiętać o tym, że aplikacje tego typu to zwykłe maszyny, które potrafią tylko interpretować i wykonywać rozkazy, zatem trzeba je dokładnie opisać i wykonywać w miarę prostych okolicznościach.

Celem tej pracy dyplomowej jest zaprojektowanie i oprogramowanie systemu *DoForMe!* umożliwiającego automatyzację zadań w systemie oraz ich planowanie na przyszłość. Aplikacja wspomagająca wspomniane zadania będzie spełniała następujące założenia funkcjonalne:

* wykonywanie skryptu Lua z określonymi zadaniami dla systemu,
* wspomaganie tworzenia skryptów za pomocą zbioru listy komend oraz sprawdzania poprawności kodu,
* możliwość automatycznego utworzenia skryptu na podstawie działań użytkownika w systemie,
* szybkie i intuicyjne zarządzanie wykonywaniem skryptów (tzw. harmonogram) z poziomu wbudowanego kalendarza,
* określanie zakresu działania "nagrywacza" skryptów (= co ma nagrywać),
* podsystem przypominający o zbliżaniu się akcji,
* definiowanie obszaru ekranu, który ma się pojawić przed kontynuowaniem skryptu.

Istnieje wiele tego typu systemów, które wspomagają automatyzację np. *AutoClickExtreme[[1]](#footnote-2)*, *WinAutomation[[2]](#footnote-3)*, *Macro Scheduler[[3]](#footnote-4)* czy *WinBatch[[4]](#footnote-5)*, jednak wszystkie te programy przedstawiają harmonogram zadań jako mało wygodną listę zadań, na której w przypadku dużej ich ilości ciężko się szuka, ponieważ zwykle nie ma żadnego sposobu grupowania. Ponadto, w większości z nich brakuje sprawdzania czy jakaś akcja w systemie została zakończona czy trwa dalej, co jest niezbędne np. w przypadku tworzenia kopii zapasowych, ponieważ czas jej tworzenia jest zależny od ilości danych.

Analiza

W tym rozdziale zostanie omówiony proces automatyzacji, czyli działania które musi poczynić użytkownik w celu zautomatyzowania zadania. Ponadto przedstawione zostaną poszczególne podsystemy i ich funkcjonalności, bez których system automatyzujący nie nadawał by się do użycia. Przybliżony zostanie m.in. sposób przechowywania zadania w programie, sposób ich wykonywania a także podsystemy umożliwiające zarządzanie harmonogramem zadań oraz powiadamianie użytkownika o zbliżających się lub zakończonych zadaniach. Oprócz tego opisane zostaną aspekty prawne dotyczące jednej z funkcjonalności, która umożliwia nagrywanie działań użytkownika a także szczegółowe różnice funkcjonalne pomiędzy istniejącymi systemami, które umożliwiają automatyzację pracy.

W celu zautomatyzowania pracy należy określić zbiór zasad i instrukcji, które musi wykonać system operacyjny. Bardzo ważnym faktem, o którym trzeba pamiętać jest to, że komputer to zwykła niemyśląca maszyna, która potrafi jedynie interpretować rozkazy, dlatego ważnym jest aby instrukcje dla systemu operacyjnego były bardzo precyzyjne i uważnie dobierane. Sposób zdefiniowania instrukcji wejściowych może mieć różną postać. Na przykład systemy *AutoClickExtreme* oraz *WinAutomation* operują na liście, na której znajdują się poszczególne akcje użytkownika[[5]](#footnote-6) [[6]](#footnote-7) (kliknięcie czy poruszenie myszką). Takie rozwiązanie jest prostsze gdy docelowa grupa użytkowników ma niewiele wspólnego z programowaniem, ponieważ użytkownik nie operuje stricte na języku programowania:

|  |
| --- |
| Sposób prezentacji zadania przez system *DoForMe!* |
| for i=1, 121, 1 do  moveTo(i, 200)  end  leftMouseClick()  for i=121, 14, -1 do  moveTo(i, 200)  end  leftMouseClick() |
| Sposób prezentacji zadania przez system *AutoClickExtreme* |
| przejrzystosc.bmp |

Jednak w przypadku bardziej doświadczonych użytkowników oznacza to brak szczegółowej modyfikacji wykonywania zadania. Na przykład w powyższym przykładzie z *AutoClickExtreme* nie jest możliwa modyfikacja sposobu ruchu kursora myszki (ani nawet podglądu po jakich współrzędnych kursor myszy się porusza)[[7]](#footnote-8). Co więcej, struktura pliku z zapisanymi akcjami jest bardziej skomplikowana niż w przypadku prostego pliku tekstowego ze skryptem, przez co trudniejsze jest odgadnięcie co robi dane zadanie nie posiadając odpowiedniego oprogramowania:

|  |
| --- |
| Wycinek pliku z zapisanym zadaniem z powyższego przykładu z *AutoClickExtreme* |
| num=0//ver=5.9  UN=000003, node=0, TypeAct=4  {  x962y501t0x961y501t265......  }  UN=000004, node=0, TypeAct=1  {  MainWindow:  exe="C:\Windows\Explorer.EXE", SbyExe=0  capt="Program Manager", class="Progman", typeCaptSearch=4 handle=10148  ChildWindow: capt="FolderView", class="SysListView32", typeCaptSearch=0 hierarchy="((0:0))", id=1, handle=1014c enable=1 Style=1442855744 exStyle=0  LimitSearch=5, PermitSetFocus=0, screen="C:\Users\jajcek\Desktop\dir\_aip\_Program Manager\ScrChild\_un=000004\_x495y427.bmp"  Borders(0, 1280, 0, 1024, 0, 1280, 0, 1024)  CLICK(xx=599, yy=501):  key=1, updown=513, dTime=171:  key=0, updown=514, dTime=94)  } |

Mimo wszystko uparty użytkownik, który ma ciut więcej doświadczenia zauważy, że można odczytać współrzędne ruchu kursora myszy zaglądając do zapisanego pliku (ciąg x962y501t0x961y501t265...), aczkolwiek nie należy to do standardowej funkcjonalności programu a tym bardziej nie jest to rozwiązanie zbyt ergonomiczne....

W przypadku gdy użytkownik nie ma doświadczenia w programowaniu skryptowym lub system oferuje zbyt mało intuicyjne zarządzanie tworzeniem zadania, możliwe jest użycie funkcjonalności jaką dają tzw. Recorder’y. Jest to Keylogger z możliwością utworzenia zadania na podstawie zarejestrowanych działań użytkownika. Takie rozwiązanie jest dostępne w bardzo wielu systemach automatyzujących (np. AutoClickExtreme[[8]](#footnote-9), WinAutomation[[9]](#footnote-10) czy Macro Scheduler[[10]](#footnote-11)). Jednak nieodpowiednie użycie Recorder’a może doprowadzić do złamania prawa. Zgodnie z kodeksem karnym, osoba nie może uruchomić oprogramowania z taką funkcjonalnością na systemie innej osoby bez jej zgody (lub wiedzy) w celu np. wykradnięcia danych czy wyrządzenia innych szkód materialnych[[11]](#footnote-12).

Po etapie utworzenia skryptu użytkownik jest już gotowy aby uruchomić swój skrypt, aczkolwiek wymaga to osobistej interwencji w systemie. W celu zautomatyzowania uruchamiania skryptu systemy takie oferują różnego rodzaju harmonogramy skryptów. W przypadku *AutoClickExtreme* została użyta prosta lista, przechowująca poszczególne zadania do których przypisana jest godzina wywołania. Jest to rozwiązanie dosyć nieczytelne w przypadku dużej ilości zadań w harmonogramie, w szczególności gdy użytkownik dodaje je nie po kolei, ponieważ oprogramowanie nie oferuje żadnego sposobu sortowania takiej listy[[12]](#footnote-13). Bardzo podobne rozwiązanie posiada system *WinAutomation*, który jednak oferuje sortowanie takiej listy względem wybranego przez nas parametru za pomocą kliknięcia nagłówka odpowiedniej kolumny*[[13]](#footnote-14)*.

W systemie *DoForMe!* został zaimplementowany całkowicie odmienny sposób zarządzania zadaniami. Do tego celu wykorzystuje się wbudowany kalendarz, na który "nakłada" się zadania na odpowiednie dni określając godzinę oraz powtórzenia. Co więcej, system oferuje powiadomienia m.in. w formie graficznej w przypadku sytuacji gdy w jednym dniu jest więcej niż jeden skrypt do wykonania o tej samej godzinie.

|  |
| --- |
| Sposób prezentacji zadań przez system *AutoClickExtreme* - brak sortowania |
| ace_task_unsorted.bmp |
| Sposób prezentacji zadań przez system *WinAutomation* - sortowanie po ostatniej kolumnie | |
| ace_task_unsorted.bmp | |

|  |
| --- |
| Sposób prezentacji zadań przez system *DoForMe!* - zadania przypisane do odpowiednich dni miesiąca; zaznaczony dzień 13 wrzesień – kolizja dwóch zadań (Job 1 oraz Job 2) |
| ace_task_unsorted.bmp |

Po utworzeniu harmonogramu skryptów użytkownik wciąż może przebywać w systemie. Dlatego niewygodne byłoby wykonywanie zadania o określonej godzinie bez uprzedniego powiadomienia o tym użytkownika. Różne systemy mają różne sposoby powiadamiania użytkownika o zadaniach z harmonogramu, które będą np. wykonane za niedługi czas, są aktualnie wykonywane lub zakończono ich wykonywanie. Jednym ze sposobów jest wykorzystanie prostego powiadomienia wyświetlonego na ekranie, które oferuje np. *WinAutomation* i które nie wymaga interwencji użytkownika[[14]](#footnote-15). Takie powiadomienie posiada jednak wadę w przypadku gdy użytkownik aktualnie pracuje na komputerze, bowiem ten sposób nie oczekuje od użytkownika żadnej odpowiedzi, dlatego gdy nadejdzie czas na uruchomienie zadania, system po prostu przejmie kontrolę nad komputerem i wykona swoje zadanie jednocześnie przerywając pracę użytkownika, aczkolwiek system umożliwia także wyświetlenie okna dialogowego, które wymaga akcji ze strony użytkownika, jednak wspomniane rozwiązanie nie jest dostępne z poziomu opcji programu a jedynie poprzez odpowiednie zdefiniowanie akcji w zadaniu do wykonania. Tego typu powiadomienie oferuje także m.in. *Macro Scheduler[[15]](#footnote-16)*. W przypadku systemu *DoForMe!* nie ma potrzeby odwoływania się do akcji/skryptu zadania, wystarczy zaznaczenie odpowiedniej opcji programu, która ma efekt globalny. Dodatkowo możliwe jest dodanie także powiadomienia dźwiękowego.

Kolejną bardzo ważną rzeczą, która jest niezbędna do w miarę inteligentnego automatyzowania zadań, jest sposób sprawdzenia stanu systemu operacyjnego, aplikacji czy innej rzeczy, którą chcemy zautomatyzować. Innymi słowy, system automatyzujący musi w jakiś sposób dowiedzieć się czy np. tworzenie kopii zapasowej danych zakończyło się aby wykonać kolejne czynności. Jest to zagadnienie dosyć złożone zważywszy na to, że każda aplikacja czy system posiadają zróżnicowane stany, które informują użytkownika o tym, że jakieś zadanie zostało zakończone. Może to być np. wyświetlenie tekstu lub obrazka albo uaktywnienie jakiegoś przycisku czy całkowite wyjście z programu. Wygodnym sposobem, zarówno dla użytkownika jak i programisty, jest pobranie fragmentu zrzutu ekranu (ustalonego przez użytkownika) i przed wykonaniem ustalonych czynności sprawdzenie czy dany fragment zawiera się w aktualnym obrazie ekranu - jeśli nie, skrypt zostanie wstrzymany i będzie mógł kontynuować tylko w przypadku gdy odnajdzie taki fragment na ekranie. Taki sposób został wykorzystany w programie *AutoClickExtreme[[16]](#footnote-17)* jak i w *DoForMe!* ze względu na jego uniwersalność oraz prostotę, aczkolwiek stawia to pewne wymogi dla użytkownika aby fragmenty, które zaznacza były unikalne oraz w miarę możliwości małe bez zbędnych elementów, w celu znalezienia danego fragmentu na ekranie w rozsądnym czasie.

|  |  |
| --- | --- |
| Gorzej | Lepiej |
| D:\DoForMe\DoForMe!\scripts\pic\pic0.bmp | D:\DoForMe\DoForMe!\scripts\pic\pic1.bmp |

Co więcej, wspomniany sposób otwiera także inną możliwość. W przypadku gdy podany fragment ekranu nie zostanie znaleziony skrypt zostanie wstrzymany i będzie mógł kontynuować dopiero wtedy gdy pojawi się na ekranie. Dzięki temu możliwe jest "naprowadzenie" (poprzez działania użytkownika) skryptu na właściwą ścieżkę, w przypadku gdy w jakichś okolicznościach się "zgubi".

Projekt

Założenia.

System jest przeznaczony dla każdego rodzaju użytkownika, od zwykłych użytkowników domowych poprzez pracowników biurowych aż do testerów oprogramowania, którzy potrzebują zautomatyzować swoją pracę w systemie lub zaplanować jakąś akcję na przyszłość bez ich późniejszej interwencji. Podczas projektowania systemu założono, że użytkownik nie potrafi programować, jednak zalecana jest podstawowa znajomość programowania skryptowego w celu wykorzystania pełnego potencjału systemu, aczkolwiek nawet pomimo braku jakichkolwiek podstaw programowania, oprogramowanie oferuje funkcjonalność, dzięki której użytkownik nie musi sam pisać kodu.

Architektura systemu jest jednowarstwowa - pracuje w obrębie systemu użytkownika bez potrzeby komunikacji z jakimkolwiek serwerem zewnętrznym lub Internetem. Ponadto dzięki braku odwołań do rejestru oraz folderów innych niż główny folder programu system jest całkowicie przenośny pomiędzy komputerami (w obrębie systemów Windows™) bez potrzeby przeinstalowywania oprogramowania. Funkcjonalność symulacji urządzeń wejścia (mysz oraz klawiatura) korzysta ze specyficznych części systemu Microsoft® Windows™, dlatego też aplikacja działa tylko na tym systemie.

dobry dla testerow oprogramowania, administratorow, progmistow, pracownikow biurowych

dobry w logowaniu, testowaniu palikacji, weryfikowaniu internetowych zasobow, zbieranie danych do reportu, processing data sets, dlugie zmudne procesy

Stworzenie podsystemu aplikacji, ktory moze byc juz testowalny

po ewentualnych zmianach w skryptach testujacych

sprawdzenie skryptu nie tylko w zakresie gramatyki ale takze logicznym i czy na pewno dobrze dziala (cza zobaczyc)

kolizja w sensie nie tylko znaczek na kalendarzu ale takze to czy nie sa za blisk siebie

Baza danych

W dzisiejszych czasach dostępnych jest kilka systemów do zarządzania bazą danych, m.in. MySQL, PostgreSQL czy SQLite, które są systemami wolnodostępnymi oraz komercyjne systemy typu Oracle, IBM DB2 czy Microsoft SQL Server. Projekt DoForMe! nie wykorzystuje intensywnie bazy danych, dlatego jako silnik do tego celu został wykorzystany SQLite. Uzasadnienie takiego wyboru wiąże się z lekkością interfejsu i tym, że nie potrzeba uruchamiać dodatkowego procesu dla serwera oraz łatwą integracją z technologią C++. Więcej informacji o samym SQLite znajduje się w rozdziale opis technologii.

W projekcie całą bazę danych tworzą trzy tabele:

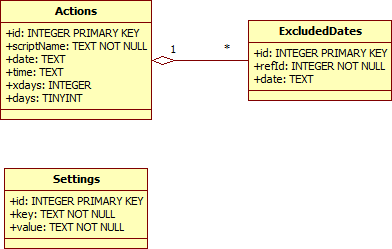


Tabela Actions przechowuje wszystkie zadania, które zostały dodane do kalendarza. W przypadku gdy zadanie posiada powtórzenia, do bazy zostaje zapisane tylko jedno zadanie z dodatkową informacją o powtórzeniach (czyli przynajmniej jedna z wartości w kolumnach xdays oraz days będzie różna od 0). Pole date przechowuje datę w formie YYYY-MM-DD, a time w HH:MM:SS. xdays określa co ile dni ma się wykonywać dane zadanie. Najbardziej skomplikowaną wartością jest tutaj days, która przyjmuje TINYINT (jest to odpowiednik char'a z C pod względem ilości bitów), aczkolwiek jest on używany jako maska bitów. Zmienna typu char składa się z 8 bitów, z których użyto 7 na dni tygodnia.

|  |  |
| --- | --- |
| Dni tygodnia i odpowiadające im bity | |
|  | Poniedziałek |
|  | Wtorek |
|  | Środa |
|  | Czwartek |
|  | Piątek |
|  | Sobota |
|  | Niedziela |

Dzięki takiemu rozwiązaniu możliwe jest proste stworzenie sumy zbioru, czyli na przykład ustalenie aby dane zadanie wykonywało się zawsze w piątki i soboty.

Innymi słowy gdy dodajemy zadanie do 1 stycznia 2013 (wtorek) i ustawiamy aby powtarzało się co 5 dni oraz w każdy piątek i sobotę, to wiersz który zostanie zapisany do bazy danych będzie miał postać (zakładamy, że id=6; pomijamy zbędne pola):

6, scriptName, '2013-01-01', time, 5, 48

Wartość 48 tworzy:

Kolejną tabelą jest tabela ExceludedDates, która jest powiązana z Actions. W opisywanym systemie istnieje funkcjonalność wykluczenia jakiejś daty z powtórzeń, np. wykonywanie zadania co 2 dni z wyłączeniem dnia 1 stycznia. Wszystkie wykluczone daty są przechowywane w ExcludedDates. Każdy wiersz tej tabeli posiada referencje do zadania w postaci id wiersza z tabeli Actions. Na przykład jeśli w powyższym przykładzie zostaną wykluczone daty 4.01.2013 oraz 11.01.2013 to do bazy ExcludedDates zostaną dodane:

1, 6, '2013-01-04'

2, 6, '2013-01-11'

Ostatnią i najprostszą tabelą jest tabela Settings, która przechowuje preferencje użytkownika. Działa ona na zasadzie prostej struktury typu klucz-wartość, gdzie w kluczu przechowujemy informacje o danej opcji programu a w wartości jej stan. Ze względu na to, że niektóre komponenty (np. checkbox i pole tekstowe) mają różne typy wartości (boolean oraz char\*) został wybrany bardziej uniwersalny typ TEXT.

[Spośród wyżej wymienionych systemów wspierających relacyjne bazy danych tylko SQLite umożliwia korzystanie z bazy danych bez dodatkowego uruchomionego procesu.] - to do opis tech.

- lua okiem programisty api

Jedną z najważniejszych możliwości języka Lua jest definiowanie własnych funkcji (których implementacje rezydują w naszym programie), które można wykorzystać podczas późniejszego pisania skryptów. Aby zdefiniować funkcję należy wykonać następujące kroki (kolejność dowolna):

1. Zdefiniować funkcję, która przyjmuje argument typu lua\_State\* i zwraca int.

2. Zarejestrować powyższą funkcję za pomocą void lua\_register(lua\_State\*, const char\*, lua\_CFunction);

Ad. 1. Po pierwsze, silnik Lua nie posiada globalnych zmiennych, a co za tym idzie, wszystkie wartości określające stan interpretera są trzymane w strukturze lua\_State. Dzięki temu silnik ma możliwość powrotu do określonego stanu[[17]](#footnote-18). Po drugie, funkcja musi zwracać wartość liczbową, która określa ile argumentów zwraca dana funkcja (argumenty są wrzucane do rejestrów za pomocą funkcji typu lua\_pushnumber(state, value)), ponieważ język Lua umożliwia zwracanie przez funkcje kilku wyników[[18]](#footnote-19), np.:

x,y = foo2()

Ad. 2. Tutaj należy zauważyć, że nazwa funkcji (która będzie nazwą widoczną w skryptach) może być różna od nazwy funkcji podczas deklaracji. Innymi słowy

lua\_register( luaState, "myFunction", pointerToFunction );

oznacza, że gdy w skrypcie napiszemy myFunction() to zostanie wywołana funkcja pointerToFunction(), która jest zdefiniowana w naszym programie.

Definicja funkcji zdefiniowana jest następująco:

int pointerToFunction( lua\_State \*state ) {

// pobierz argumenty z tablicy Lua i zrob cos z nimi

// zwroc informacje o tym ile argumentow zwrocila dana funkcja

return 0;

}

Interpreter języka od wersji 5.0 bazuje na rejestrach, które są umieszczane na stosie w tzw. activation records. Gdy dochodzi do momentu interpretowania zdefiniowanej przez programistę funkcji, na stosie zostaje przydzielona pamięć na activation record na tyle duża by pomieścić przebywające tam rejestry, które przechowują argumenty funkcji. Zatem aby ich użyć należy je uprzednio pobrać z rejestrów:

int \_arg1 = ( int )lua\_tonumber( state, -2 );

int \_arg2 = ( int )lua\_tonumber( state, -1 );

drugim argumentem funkcji jest indeks elementu, który chcemy pobrać z rejestru. Rejestr w tym przypadku najlepiej traktować jako zwykły stos, na który są odkładane argumenty od lewej do prawej. Dlatego kolejność "ściągania" argumentów jest odwrotna do kolejności "wrzucania". Wartość oznacza wierzchołek stosu. Im ta wartość jest mniejsza tym niższy na stosie pobieramy argument. Zauważmy, że nie musimy dbać o kolejność pobierania argumentów (w powyższym kodzie najpierw pobieramy argument, który znajduje się najniżej; domyślnie zwracamy ją jako liczbę double), ważne jest tylko podanie prawidłowego indeksu. Istnieją oczywiście inne wersje funkcji pobierającej daną z rejestru dla innych typów, np. lua\_tostring( lua\_state, int );, która zwraca const char\*.

-- wykonywanie instrukcji skryptu LuaEngine

Bardzo ważnym faktem (i oczywistym) silnika Lua jest to, że gdy skrypt zostanie wczytany, to po jego uruchomieniu silnik wykonuje każdą instrukcję po kolei (w sensie tak jak została napisana logika skryptu). Jest to logiczne i wydaje się normalne zachowanie, aczkolwiek w pewnych przypadkach może to być pewnym utrudnieniem.

Przykładem, na którym zostanie omówione wspomniane utrudnienie będzie prosty skrypt, który wykonuje dwa kliknięcia myszką. Pierwszym kliknięciem zostaje wciśnięty przycisk, który wyświetla jakieś okno dialogowe, za to drugim kliknięciem zostanie wciśnięty inny przycisk ale już na nowo otwartym oknie. Skrypt może wyglądać np. tak:

leftClickAt( 100, 100 )

leftClickAt( 200, 200 )

Powyższy skrypt zrobi dokładnie to co zostało dla niego zdefiniowane (czyli kliknie dwa razy w miejscach o podanych współrzędnych), jednakże może się zdarzyć, że po pierwszym kliknięciu okno dialogowe może nie zdążyc się pojawić przed drugim kliknięciem. Jedno z rozwiązań samo się narzuca - trzeba dodać jakieś uśpienie pomiędzy tymi dwoma kliknięciami, aby okno dialogowe miało czas na wyświetlenie się. Innym sposobem jest bardziej zaawansowany podsystem, który poszukuje ustalonego przez użytkownika fragmentu ekranu tak jak zostało to opisane przy rozdziale analizy zagadnienia. Jednak w tym podrozdziale uwaga zostanie skupiona wokół uśpienia, ponieważ wymagało to odpowiedniego zaprojektowania takiego silnika, a w przypadku pobierania fragmentu ekranu jest to po prostu funkcja udostępniania użytkownikowi.

Istnieje wiele sposobów na rozwiązanie takiego uśpienia, zostanie tutaj opisane część z nich, które zostały zaimplementowane w systemie DoForMe!.

-- nowa komenda - sleep

Pierwszym sposobem, jaki może się narzucić na myśl, jest nowa komenda, który zatrzymuje wykonanie skryptu w określonym miejscu na określony czas. Rozwiązanie takie daje szereg zalet:

* precyzja - wiadomo dokładnie gdzie i na ile zostanie zatrzymane wykonanie skryptu
* modyfikowalność - bez problemu można dodać lub usunąć uśpienie w konkretnym miejscu
* prostota

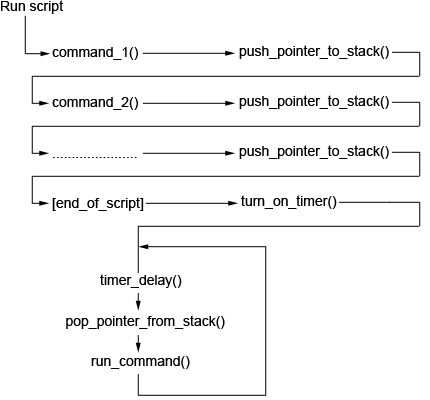
aczkolwiek są także i wady:

* w przypadku gdy jest dużo komend, które muszą być wywoływane w pewnych odstępach czasu, trzeba dodać dużo uśpień, przez co traci się na przejrzystości kodu
* zwiększenie objętości kodu

-- nowa opcja - odstep czasowy co kazda komende

Kolejnym pomysłem może być parametryzowalny odstęp czasowy co każdą instrukcję w taki sposób aby nie było potrzeby umieszczania co każdą komendę uśpienia. Aby osiągnąć taki efekt trzeba poczynić pewne przygotowania w przypadku wykonywania instrukcji przez silnik Lua. Przy okazji zostanie wytłumaczony sposób przechowywania implementacji poleceń, które zostały udostępnione użytkownikowi.

Problem ten został rozwiązany za pomocą utworzenia nowego stosu, na który odkładane są poszczególne komendy. Sposób wykonywania skryptu przez interpreter Lua pozostaje bez zmian, zmiana jest dopiero w przypadku implementacji poszczególnych komend. Jeszcze przed problemem "uśpienia", implementacje funkcji dostępnych w skryptach miały bezpośrednio zaimplementowany kod akcji, który dana komenda miała wykonać. Zmiana dotyczy tego, że teraz każde udostępnione użytkownikowi polecenie realizowane jest za pomocą dwóch oddzielnych funkcji, z których pierwsza tylko wrzuca wskaźnik na stos do drugiej funkcji, która faktycznie posiada implementację danego polecenia. Gdy Lua zinterpretuje już cały skrypt to wtedy zostanie uruchomiony licznik, który co jakiś czas ściąga po jednej komendzie ze stosu i ją wykonuje. Sposób ten został zaprezentowany na rysunku poniżej:



Dzięki takiemu podejściu czas pomiędzy komendami (timer\_delay()) jest wartością parametryzowalną, zatem możliwa jest jego zmiana nawet podczas wykonywania skryptu. Ta właściwość zostanie później wykorzystana. [Więcej informacji na temat organizacji struktury kodu podamy przy omawianiu klasy LuaApiEngine.] Ważnymi zaletami takiego rozwiązania to są m.in.:

* parametryzowalny czas pomiędzy wykonaniami komend,
* uśpienie jest niewidoczne z poziomu kodu, dzięki temu kod jest przejrzysty,
* od strony programistycznej możemy w łatwy sposób filtrować i modyfikować komendy ściągane ze stosu.

oraz wady:

* więcej roboty od strony programistycznej - trzeba odpowiednio zaprojektować strukturę kodu,
* dla każdej komendy potrzeba dwóch funkcji - jedna funkcja, która wrzuca na stos wskaźnik do drugiej funkcji
* uśpienie bazuje na wartości ustawionej w opcjach programu, zatem po przeniesieniu skryptu na inny komputer może się on inaczej zachowywać (jednak problem ten może zostać rozwiązany za pomocą nowej komendy, która zostanie teraz przedstawiona).

-- nowa komenda - setInterval(int)

Pomysł ten bazuje głównie na rozwiązaniu z poprzedniego rozdziału. Jak już wcześniej zostało wspomniane, czas po którym jest wykonana następna komenda jest wartością parametryzowalną, którą można zmienić nawet podczas działania skryptu. Dzięki temu bardzo łatwo wykorzystać ten fakt wprowadzając nową komendę, która faktycznie będzie zmieniała opóźnienie czasowe podczas działania skryptu. Innymi słowy po napisaniu kodu:

setInterval(1000)

command1()

command2()

command3()

pomiędzy wszystkimi komendami, które są wywoływane po funkcji setInterval() będzie odstęp czasowy równy 1 sekundzie.

Zaletami takiego sposobu są:

* w sytuacji, gdy w jakimś miejscu (ale nie wszędzie) powinno występować wiele uśpień pomiędzy komendami, możemy je zastąpić jedną komendą setInterval(int),
* w połączeniu ze sleep(int) daje komfortowe narzędzie do manipulowania opóźnieniami,
* dzięki istnieniu funkcji w kodzie staje się on przenośny (nie tak jak w przypadku opcji programu omówionej w poprzednim rozdziale).

[[to jest jak sleep wiece bez sensu]-- nowy argument dla kazdej komendy

Istnieje także metoda (która nie została zaimplementowana w programie), aby do każdej funkcji w skrypcie przekazywać dodatkowy argument informujący po jakim czasie ma się wykonać komenda. Zaletą takiego rozwiązania może być:

* dokładne i przejrzyste wskazanie, przy której komendzie zostanie wstrzymany skrypt

Jednak istnieje parę wad, które przeważyły o nie implementowaniu tego w programie:

* dużo niepotrzebnego kodu,
* brak elastyczności od strony programistycznej - gdy nie zastosuje się tego sposobu od samego początku, w późniejszej fazie jest bardzo dużo do zmiany (trzeba zmienić każdą funkcję odpowiadającą za komendę),
* w przypadku istnienia poprzednich rozwiązań, to zdaje się nieużyteczne.]

-- Kalendarz

Do stworzenia harmonogramu dla zadań został użyty kalendarz wbudowany w bibliotekę Qt. W celu dostosowania go do wymaganych potrzeb została nadpisana funkcja rysująca komórkę kalendarza. Zostało to bardzo wygodnie zaprojektowane (od strony Qt), ponieważ taka funkcja przyjmuje jako argumenty datę oraz współrzędne prostokąta przy odrysowywaniu poszczególnych komórek[[19]](#footnote-20).

Rozwiązanie takie w porównaniu do zwykłych list zadań, cechuje się paroma zaletami, m.in. to, że bez żadnego wysiłku otrzymujemy grupowanie zadań po dniach oraz umożliwia większą czytelność i bardziej intuicyjną modyfikację istniejących zadań, ponieważ powtórzenia także są widoczne na kalendarzu. Dzięki temu, że kalendarz wyświetla wybrany przez użytkownika miesiąc, możliwe jest utworzenie skończonej liczby powtórzeń zadania, które będą pokrywać tylko widoczny miesiąc. W przypadku listy może to być uciążliwe, ponieważ użytkownik zazwyczaj widzi całą listę, aczkolwiek możliwe jest stworzenie czegoś w rodzaju filtrowania po miesiącach. Porównanie harmonogramów dla zadań znajduje się na rysunkach bla bla w rozdziale bla bla.

Mimo, że większą część kalendarza dostajemy za darmo z Qt to podstawą jest odpowiednie dobranie struktury przechowującej dane o zadaniach. Do takiego harmonogramu zostały dodane dwie struktury typu klucz-wartość, w których kluczem jest data a wartością lista przechowująca zadania dla konkretnej daty. Jedna z map przechowuje wszystkie zadania, które zostały dodane do kalendarza, nie uwzględniając ich powtórzeń, dlatego staje się ona bazą dla drugiej mapy, które przechowuje jedynie powtórzenia, aczkolwiek ograniczając je tylko do aktualnie wybranego przez użytkownika miesiąca. Dlatego też pierwsza z map jest tworzona tylko podczas wczytywania zadań z bazy danych oraz uaktualnia się w momencie dodawania nowych, jednak nie ulega zmianie podczas zmiany miesiąca widocznego na kalendarzu w przeciwieństwie do drugiej mapy, która musi uaktualnić powtórzenia dla nowo wyświetlonego miesiąca.

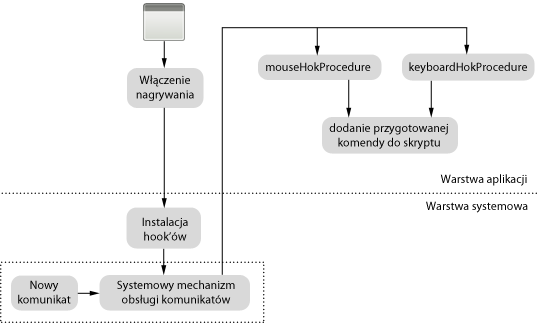
Ze względu na to, że podczas wyświetlania danego miesiąca użytkownik jest w stanie zobaczyć parę dni pochodzących od innych miesięcy aktualna implementacja wylicza powtórzenia dla jednego poprzedniego oraz następnego miesiąca, aczkolwiek da się to zoptymalizować aby wyliczało tylko dla maksymalnej liczby widocznych dni z innego miesiąca.

Liczba ta jest równa 14, ponieważ kalendarz tworzy szachownicę rozmiaru 7x6 (pominięto pola na nazwy dni tygodnia oraz numer tygodnia), czyli 42 pola. Gdy pola zostaną wypełnione od samego początku dniami pochodzącymi z nieprzestępnego lutego (czyli z miesiąca, który posiada najmniej dni aby uwidocznić jak najwięcej dni innych miesięcy) otrzymamy 42-28=14 wolnych pól.

-- modul recorder

Moduł ten umożliwia przetwarzanie działań użytkownika na odpowiednie komendy, które następnie dodawane są do okna z kodem skryptu. Działa w całości na podstawie przechwytywanych komunikatów systemowych. W trakcie nagrywania inicjalizowany jest także licznik, który oblicza ile czasu minęło pomiędzy dwoma komunikatami, które zostały przekształcone w komendy do skryptu. Po każdym dodaniu komendy do skryptu licznik jest zerowany i zaczyna liczyć od początku aż do pojawienia się następnego komunikatu. Implementacja modułu opiera się na działaniu hook'ów systemowych niskiego poziomu podpiętych pod klawiaturę oraz mysz, które zostają włączone w zależności od ustawień użytkownika. [W ramach uproszczenia użyto właśnie tych do implementacji, ponieważ drugi rodzaj hook'ów "wstrzykuje" (ang. DLL injection) się do procesów, dlatego te drugie muszą istnieć w dwóch wersjach - dla 32 oraz 64-bitowych procesów oraz ich kod musi rezydować w bibliotekach DLL.] - to do opisu hookow

W momencie nadejścia nowego komunikatu zostaje on przekazany do odpowiedniej funkcji (w tym przypadku keyboardHookProcedure w przypadku komunikatu pochodzącego od klawiatury i mouseHookProcedure w przypadku myszy), która została zdefiniowana przy tworzeniu hook'a. Przekazywaniem komunikatów w odpowiednie miejsce zajmuje się mechanizm systemowy odpowiadający za obsługę komunikatów, w którym uprzednio został zainstalowany hook. Po otrzymaniu przez funkcję danego komunikatu rozpoczyna się proces przefiltrowywania jego treści aby np. pobrać kod wciśniętego klawisza lub sprawdzić który przycisk myszy został wciśnięty i na jakich współrzędnych. Na podstawie tych informacji moduł przygotowuje odpowiednią komendę, którą wysyła prosto do okna zawierającego skrypt. Schemat działania (bez uwzględnienia licznika) został zaprezentowany na rysunku blabla.



-- screensaver

Podsystem ten umożliwia użytkownikowi zaznaczenie fragmentu ekranu, który jest następnie zapisywany do pliku jako bitmapa i wykorzystywany do wyszukiwania go na aktualnym ekranie. Algorytmem, który znajduje obrazek w innym obrazku jest prosty template matching, który działa na podstawie dokładnego sprawdzania wartości RGB pikseli. Funkcjonalność ta zostaje uruchomiona w momencie wykonania odpowiedniej kombinacji klawiszy w trakcie nagrywania skryptu za pomocą modułu Recorder. W momencie przejścia w tryb zaznaczania fragmentu ekranu niezbędne jest zignorowanie przetwarzania komunikatów przez system Recorder (czyli przekazanie komunikatów dalej nic z nimi nie robiąc), ponieważ w sytuacji gdy użytkownik nagrywania działania myszką to przy zaznaczaniu fragmentu ekranu (który wymaga działania za pomocą myszy) jednocześnie zapisze swoje ruchy i kliknięcia do skryptu. Moduł ten reaguje na komunikat zwolnienia lewego przycisku myszy, który przywraca pierwotne działanie Recorder'a, tym samym zapisując zaznaczoną bitmapę do pliku oraz tworząc odpowiednią komendą wstrzymującą działanie skryptu w przypadku braku bitmapy na ekranie. Jest to wygodne rozwiązanie, ponieważ nie zmusza użytkownika do kliknięcia kolejnej kombinacji klawiszy aby powrócić do normalnego trybu, aczkolwiek w przypadku gdy chce zaznaczyć więcej fragmentów musi powtórzyć przejście do trybu zaznaczania fragmentu ekranu, jednak zaleca się używanie jednego unikalnego obrazka w skali ekranu ze względu na wydajność znajdowania obrazka.

Jak zostało wcześniej powiedziane, po przejściu do opisywanego trybu użytkownik musi wykonać pewną kombinację klawiszy na klawiaturze. Taki zabieg rodzi kolejny problem - zanim użytkownik naciśnie wszystkie przyciski z danej kombinacji moduł Recorder'a nagra część z tych klawiszy, mimo że ich nie powinno być. Dlatego po odczytaniu specjalnej kombinacji należy ją usunąć ze skryptu.

Implementacja rozwiązania polega na zrobieniu zrzutu aktualnego stanu ekranu i stworzeniu nowej formatki pokrywającej cały ekran, na której jest wyrysowywany uprzednio zrobiony zrzut ekranu. Dopiero po tych etapach użytkownik ma możliwość zaznaczenia danego fragmentu ekranu, czyli tak naprawdę fragmentu bitmapy, którą jest zrzut ekranu.

-- cos o hookach

Hooking jest terminem dosyć rozległym, który w dużym skrócie oznacza między innymi przechwytywanie komunikatów systemowych. Dzieli się m.in. na sposoby w jakie zostaje zainstalowany w systemie oraz na zasięg przechwytywanych informacji. Technika ta jest stosowana głównie do szpiegowania, lub łagodniej mówiąc, pobierania pewnych informacji, które są głębiej zakorzenione w aplikacji. Nie oznacza to, że takie dane są zawsze tajne i ich odczytanie jest przestępstwem. Dane takie służą bardzo często do dokładniejszego przestudiowania danej aplikacji jeśli dokumentacja techniczna jest mocno wybrakowana lub w ogóle jej nie ma. Za pomocą hook'ów możliwe jest przechwytywanie komunikatów jakie są wysyłane do aplikacji w odpowiedzi na jakieś zdarzenie, dzięki temu można łatwo przefiltrować lub po prostu odczytać ich zawartość. W niniejszej pracy uwaga zostanie skupiona głównie na przechwytywaniu komunikatów systemowych za pomocą niskopoziomowych hook'ów ze względu na to, że właśnie to jest wykorzystywane w projekcie. - dalej jest w texie

-- cos o Qt

Framework, który posiada bardzo bogaty wachlarz możliwości. Służy nie tylko do tworzenia GUI, ale posiada także dobrze rozbudowany odpowiednik biblioteki STL z paroma swoimi dodatkami oraz m.in. obsługę sieci, OpenGL, XML, SVG czy multimediów. Zawiera dużo gotowych komponentów, które w połączeniu z wizualnym edytorem okna aplikacji dają duże możliwości. Został wykorzystany m.in. w środowisku KDE[[20]](#footnote-21), Autodesk Maya, VirtualBox oraz Mathematica. Cechami szczególnymi są m.in.:

* przenośność (nawet w obrębie systemów wbudowanych) - obsługuje m.in. Windows, Mac, Linux, Solaris, Embedded Windows, Embedded Linux,
* idea Signal-Slot
* dodatkowy kompilator tzw. moc (Meta Object Compiler)
* brak kompatybilności wstecz (jeśli różni się pierwsza cyfra, tzn. 4.0 i 4.1 są kompatybilne, ale 3.0 z 4.0 już nie)

Idea signal-slot (inaczej sygnał-gniazdo) jest fundamentem dla komunikacji pomiędzy obiektami w aktualnej wersji frameworka. Zastępuje ona koncepcję zdarzeń w aplikacji, która zazwyczaj opiera się na funkcjach zwrotnych (tzw. callback'ów), czyli wskaźników do funkcji, które są przekazywane jako argument do innej funkcji przetwarzającej dane zdarzenie.

Rozwiązanie bazujące na wspomnianych callbackach nie jest type-safe, czyli nigdy nie można być pewien co tak naprawdę zostało przekazane do funkcji zwrotnej. Obrazuje to poniższy przykład, który się skompiluje, aczkolwiek wystąpi błąd w czasie działania programu:

#include <cstdio>

#include <string>

#include <iostream>

typedef void( \*MyCallback )( std::string\* );

void myCallback( std::string\* a ) {

if( a )

std::cout << a->c\_str();

}

void ourProcessingFunc( MyCallback myCallback, void\* someDataForCallback ) {

myCallback( ( std::string\* )someDataForCallback );

}

int main({

char \*someData = "a text";

ourProcessingFunc( myCallback, someData );

return 0;

}

Oczywiście odnosi się to nie tylko do callback'ów, ale ogólnie do funkcji w C++. W przypadku sygnałów i gniazd, sygnatura callback'a (funkcji zadeklarowanej jako gniazdo) musi odpowiadać sygnaturze funkcji przetwarzającej zdarzenie (funkcja zadeklarowana jako sygnał), dzięki temu nie ma możliwości przekazania jej parametrów nieodpowiedniego typu, ponieważ wystąpi błąd kompilacji. Przykład ilustruje listing X (clicked() oraz funcToHandleButtonClick() mają takie same sygnatury):

conncet( ButtonClass, SIGNAL( clicked() ), MainAppClass, SLOT( funcToHandleButtonClick() ) )

Dokumentacja Qt deklaruje, że callback'i są mocno związane z funkcjami przetwarzającymi zdarzenie, ponieważ takie funkcje muszą wiedzieć, którą dokładnie funkcję zwrotną chcemy wywołać. Aczkolwiek wygląda to bardziej na zabieg marketingowy, ponieważ w dużej większości implementacji, do funkcji przetwarzającej zdarzenie przesyła się jako argument wskaźnik do innej funkcji, która pełni rolę callback'a[[21]](#footnote-22), dlatego możliwe jest przesłanie jej dowolnej funkcji, która spełnia określoną sygnaturę.

Aby możliwe było użycie nowych własności jakie wprowadza Qt, niezbędny jest dodatkowy kompilator, który przekształca makra pochodzące z Qt na kod C. Jest nim MOC (Meta Object Compiler), który umożliwia między innymi użycie wspomnianej idei sygnałów i gniazd.

-- cos o SQLite

Jest to system do zarządzania relacyjną bazą danych. Bazy takie służą do przechowywania danych w dużych tabelach, z których można pobierać dane na podstawie zapytań SQL. W odróżnieniu od innych systemów, SQLite jest bardzo prosty w przypadku złożoności administrowania bazą oraz nie wymaga dużych zasobów komputerowych, przez co jest używany m.in. w systemach mobilnych typu Android[[22]](#footnote-23) czy iOS[[23]](#footnote-24).

Głównymi cechami tego systemu są m.in. brak dodatkowego procesu dla serwera, innymi słowy nie uruchamia się zewnętrznego serwera w celu używania bazy danych, lecz wystarczy kompilacja projektu z dołączonym źródłem dla SQLite. Dzięki takiemu rozwiązaniu nie zmusza się użytkownika do konfiguracji serwera, ponieważ go nie ma. Co więcej, system ten jest w pełni przenośny, ponieważ istnieją implementacje dla różnych przenośnych języków programowania. Ponadto dzięki temu, że system egzystuje w postaci kodu, jest on bardzo prosty do przystosowania go do projektów. Dla przykładu SQLite dla C/C++ rezyduje w dwóch plikach - sqlite3.h oraz sqlite3.c, zatem wystarczy proste dodanie plików do projektu i jego skompilowanie. Kolejną zaletą tego systemu jest to, że wszystkie dane wrzucane do bazy danych znajdują się w pojedynczym pliku, dzięki temu wszystko jest w jednym miejscu i jest łatwe do przenoszenia (oczywiście jest możliwość utworzenia pliku dla każdej tabeli z osobna).

Ze względu na prostotę zarządzania oraz łatwość przystosowania go do projektów, został on użyty w także projekcie DoForMe!.

-- cos o template matching

Jest to technika wykorzystywana do wyszukiwania obrazu w obrazie. Wydawać się to może prostym zagadnieniem gdy przeszukiwanie obrazu wykonuje się piksel po pikselu, aczkolwiek staje się ono trudniejsze gdy zostaną wzięte pod uwagę czynniki takie jak szum obrazu oraz tolerancja (czyli jak maksymalnie mogą się różnić obrazy aby można było stwierdzić, że są "takie same"). Mimo, że technika ta rozciąga się na różne poziomy zaawansowania to jednak na potrzeby projektu DoForMe! nie trzeba było wykorzystywać bardziej zaawansowanych algorytmów, ponieważ system nie wykorzystuje obrazów utworzonych np. za pomocą aparatu czy skanera a jedynie zrzuty ekranu.

Dlatego też do wyszukiwania fragmentu ekranu został wykorzystany naiwny algorytm, który próbuje dopasować szukany obrazek przesuwając go o piksel w pionie i w poziomie. Pseudokod został zaprezentowany w listingu X:

1. DENIS SAFONOV, *AutoClickExtreme*, <http://www.autoclickextreme.com/>, 2010 [↑](#footnote-ref-2)
2. SOFTOMOTIVE SOLUTIONS LTD, *WinAutomation*, <http://www.winautomation.com/>, 2012 [↑](#footnote-ref-3)
3. MJT NET LTD, *Macro Scheduler*, <http://www.mjtnet.com/>, 2012 [↑](#footnote-ref-4)
4. WILSON WINDOWWARE, *WinBatch*, <http://winbatch.com/>, 2012 [↑](#footnote-ref-5)
5. DENIS SAFONOV, *Task Manager help*, <http://www.autoclickextreme.com/help_en/tasker.htm>, 2012 [↑](#footnote-ref-6)
6. SOFTOMOTIVE SOLUTIONS LTD, *Visual Designer*, <http://www.winautomation.com/features.html>, 2012 [↑](#footnote-ref-7)
7. DENIS SAFONOV, *AutoClickExtreme - "Change of Event" Window*, <http://www.autoclickextreme.com/help_en/topic3.htm>, 2012 [↑](#footnote-ref-8)
8. DENIS SAFONOV, *AutoClickExtreme - Simple View*, <http://www.autoclickextreme.com/help_en/topic1.htm>, 2012 [↑](#footnote-ref-9)
9. SOFTOMOTIVE SOLUTIONS LTD, *Macro Recorder*, <http://www.winautomation.com/features.html>, 2012 [↑](#footnote-ref-10)
10. SOFTOMOTIVE SOLUTIONS LTD, *Macro Scheduler – Recording Macros s. 15,* <http://www.mjtnet.com/MacroSchedulerManual.pdf>, 2012 [↑](#footnote-ref-11)
11. Ustawa z dnia 6 czerwca 1997 r. Kodeks karny (Art. 267 §1, §2, §3 oraz 269b §1) [↑](#footnote-ref-12)
12. DENIS SAFONOV, *Task Manager help*, <http://www.autoclickextreme.com/help_en/tasker.htm>, 2012 [↑](#footnote-ref-13)
13. sprawdzane na wersji 4.0.2 [↑](#footnote-ref-14)
14. ADMIN OF WINAUTOMATION'S BLOG, *Display Message VS Display Notification*, <http://www.winautomation.com/blog/display-message-vs-display-notification>, 2010 [↑](#footnote-ref-15)
15. MJT NET LTD, *Macro Scheduler - Command Reference*, <http://www.scribd.com/doc/36237955/Macro-Scheduler-Help#outer_page_99>, 2010 [↑](#footnote-ref-16)
16. DENIS SAFONOV, AutoClickExtreme Help - Pixel Control, http://www.autoclickextreme.com/help\_en/sc\_PixControl.htm [↑](#footnote-ref-17)
17. ROBERTO IERUSALIMSCHY, *Reference Manul*, <http://www.lua.org/manual/5.2/manual.html#lua_State>, 2012 [↑](#footnote-ref-18)
18. ROBERTO IERUSALIMSCHY, *Multiple results*, <http://www.lua.org/pil/5.1.html>, 2012 [↑](#footnote-ref-19)
19. http://doc.qt.digia.com/qt/qcalendarwidget.html#paintCell [↑](#footnote-ref-20)
20. http://en.wikipedia.org/wiki/Qt\_(framework)#cite\_note-11 [↑](#footnote-ref-21)
21. scott meyers more effective c++ [↑](#footnote-ref-22)
22. http://developer.android.com/training/basics/data-storage/databases.html [↑](#footnote-ref-23)
23. https://developer.apple.com/technologies/ios/data-management.html [↑](#footnote-ref-24)