Interfejs użytkownika w aplikacjach naukowych, obliczeniowych i inżynierskich

Małgorzata Płotka
Paweł Syty
Politechnika Gdańska
Wydział Fizyki Technicznej i Matematyki Stosowanej
Katedra Fizyki Teoretycznej i Informatyki Kwantowej
ul. Narutowicza 11/12, 80-233 Gdańsk
mplotka@mif.pg.gda.pl

Kansei 2009
Interfejs użytkownika – Kansei w praktyce
ISBN 978-83-89244-78-9

Streszczenie
Tekstowy interfejs użytkownika jest jeszcze dość powszechnie spotykany w aplikacjach naukowych, obliczeniowych oraz inżynierskich. Można zaobserwować pewną niechęć użytkowników tego typu aplikacji (naukowców, szczególnie w dziedzinie nauk ścisłych) do używania interfejsów graficznych (GUI), nawet gdy są one dostępne dla danego narzędzia. W naszej pracy szczegółowo analizujemy to zjawisko, zarówno pod kątem technicznym jak i psychologicznym. Ponadto przeprowadziliśmy sondę, w której zbadaliśmy, jakie warunki powinien spełniać GUI dla tej specyficznej grupy użytkowników, aby miał szansę zastąpić przestarzały, tekstowy interfejs. Wyniki sondy oraz własne doświadczenia są podstawą przedstawionej analizy procesu inżynierii wymagań w przypadku definiowania GUI dla aplikacji naukowych, zilustrowanej konkretnymi przykładami.

Wstęp
Istnieje wiele publikacji na temat interfejsów aplikacji webowych, czy też ogólnie interfejsów graficznych (Spolsky, 2001; Johnson, 2000). Jednak, dla kontrastu, niewiele publikacji poświęconych jest tematyce interfejsów aplikacji naukowych, obliczeniowych i inżynierskich (dla uproszczenia, w dalszej części publikacji będą one nazywane po prostu aplikacjami

This paper should be cited as: Płotka, M., & Syty, A. (2009). Interfejs użytkownika w aplikacjach naukowych , obliczeniowych i inżynierskich. Proceedings of the Conference: Interfejs użytkownika - Kansei w praktyce (pp. 74–82). Warsaw: Wydawnictwo PJWSTK.

bądź programami naukowymi), a przecież nie można zapomnieć, że na potrzeby nauki także tworzy się szereg programów użytkowych. Jaki zatem interfejs użytkownika widzi najchętniej naukowiec w oprogramowaniu, którego na co dzień używa? Co decyduje o tym, że prosty interfejs tekstowy (najczęściej oparty o komendy i polecenia wierszowe, przetwarzanie wsadowe, system pytanie-odpowiedź lub proste formularze) jest znacznie częściej wybierany przez naukowców niż nowoczesny interfejs graficzny, który może składać się m.in. z okien dialogowych, menu czy też systemu manipulacji bezpośredniej? Niewątpliwie, wiele tu zależy od rodzaju aplikacji i operacji za jej pomocą wykonywanych, jednak pewne, opisane w kolejnych rozdziałach prawidłowości można wyraźnie zaobserwować. Jak zatem powinien wyglądać proces inżynierii wymagań w przypadku definiowania interfejsu użytkownika dla takiego typu programów? Uznaliśmy, że warto się bliżej przyjrzeć wszystkim tym zagadnieniom. W niniejszej pracy staramy się odpowiedzieć na postawione pytania na podstawie własnych doświadczeń oraz przeprowadzonych analiz. Dodatkowo, przedstawiamy wyniki własnych badań dotyczących konstruowania GUI dla programów obliczeniowych. Wnioski są często bardzo zaskakujące.

Interfejs tekstowy a interfejs graficzny

Interfejs tekstowy, realizowany np. poprzez linię komend, powszechnie uważany jest za dość archaiczny. Do dziś używany jest przez niewielka grupę użytkowników zaawansowanych, specjalistów, szczególnie tam, gdzie mamy do czynienia z zamkniętymi zadaniami. Osoby go używające chwalą ten typ interfejsu przede wszystkim za precyzję, skuteczność oraz szybkość działania. Niewątpliwie, dla osób mniej zaawansowanych może on się okazać trudny do nauki, ponieważ wymaga zapamiętania szeregu poleceń, komend. Przed przystąpieniem do pracy (przy pierwszym użyciu programu albo po długiej przerwie w jego używaniu), lub też w celu przypomnienia/poznania nowej komendy, konieczne jest sięgnięcie do nierzadko bardzo obszernej dokumentacji, w której opisane są wszystkie funkcje programu.

Interfejs graficzny (ang. ***G**raphical **U**ser **I**nterface*, **GUI)** został po raz pierwszy zaproponowany i zastosowany w praktyce przez firmę Xerox w latach 70-tych XX wieku (Akass, 2001). Choć nadal wiele programów posiada tylko tekstowy interfejs, to z czasem GUI zaczął je wypierać. Stało się tak głównie dzięki szybkiemu rozwojowi środowisk graficznych umożliwiających użytkownikowi komunikację z komputerem w trybie graficznym, w którym programista może bezpośrednio sterować pojedynczymi pikselami na ekranie. Do najpopularniejszych interfejsów tego typu należą te realizowane za pomocą okienek dialogowych bądź menu użytkownika (główne, znajdujące się w pasku narzędzi; pomocnicze - pod prawym przyciskiem myszy itp.). Użytkownicy chwalą sobie m.in. ich prostotę, przejrzystość funkcji, łatwość i zrozumiałość użycia oraz stosunkowo krótki czas nauki. Jest to więc idealny typ interfejsu dla popularnych programów, powszechnie używanych przez najróżniejsze grupy użytkowników. W programach mniej popularnych (specjalistycznych), GUI pozwala na zapoznanie się z podstawową funkcjonalnością aplikacji przez osoby, które dopiero zaczynają z nią pracować. Jednak wraz z „oswojeniem" się z programem, zaawansowani użytkownicy często rezygnują (o ile to tylko możliwe) z używania udogodnień, jakie daje interfejs graficzny, w celu podniesienia ergonomii

swojej pracy oraz zwiększenia czytelności wykonywanych zadań polegających np. na wprowadzeniu danych.

Przykłady interfejsów tekstowych w aplikacjach naukowych

Od momentu upowszechnienia się komputerów, powstała ogromna liczba różnego rodzaju programów obliczeniowych, naukowych i inżynierskich. Można wśród nich znaleźć zarówno programy płatne jak i darmowe, stworzone przez duże korporacje jak i rozwijane przez grupki zapaleńców, dodatkowo, czasami udostępniany jest ich kod źródłowy. Część z nich jest przeznaczona dla bardzo wąskiej grupy specjalistów, część jest programami ogólnego zastosowania. Mimo rozpowszechnienia się GUI, to znakomita większość tych programów posiada ciągle tekstowy interfejs użytkownika (jako jedyny sposób komunikacji użytkownika z programem, bądź jako alternatywę do GUI).

Przykładem jednego z najprostszych interfejsów użytkownika jest ten użyty w programach obliczeniowych fizyki i chemii kwantowej: MOLPRO (Rys. 1), JMATRIX (Rys. 2; Syty, 1999) oraz GRASP2. Interakcja użytkownika z tymi programami jest ograniczona do edycji pliku tekstowego, zawierającego instrukcje i parametry wczytywane przez właściwy program. Użytkownik zwykle musi dokładnie znać strukturę tego pliku, w przypadku jakiejkolwiek pomyłki program zgłosi błąd bądź wykona niepoprawne obliczenia.

```
***,Rb2

 !memory,80,m
GPRINT,BASIS,DISTANCE,ANGLES,ORBITAL,CIVECTOR,PAIRS,CP,CS,REF,PSPACE,MICRO,VARIA
BLE,OPTIONS,THRESHOL,ZMATRIX,MPPERR,OPERATOR,EXTENSIO;

rvec=[80.]
![70.,68.,66.,64.,62.,60.,58.,56.,54.,52.,50.,48.,46.,44.,42.,40.,38.,36.,34.,32
.,30.,28.,26.,24.,22.,20.,19.,18.,17.,16.,15.,14.4,14.,13.5,13.,12.5,12.,11.5,11
.,10.5,10.,9.5,9.,8.75,8.5,8.25,8.,7.75,7.5,7.25,7.,6.8,6.6,6.4,6.2,6.,5.8,5.6,5
.4,5.2,5.,4.8,4.6,4.4,4.2,4.,3.8,3.6,3.4,3.2,3.]
![90.,88.,86.,84.,82.,80.,78.,76.,74.,72.,70.,68.,66.,64.,62.,60.,58.,56.,54.,52
.,50.,48.,46.,44.,42.,40.,38.,36.,34.,32.,30.,28.,26.,24.,22.,20.,19.,18.,17.,16
.,15.,14.4,14.,13.5,13.,12.5,12.,11.5,11.,10.5,10.,9.5,9.,8.75,8.5,8.25,8.,7.75,
7.5,7.25,7.,6.8,6.6,6.4,6.2,6.,5.8,5.6,5.4,5.2,5.,4.8,4.6,4.4,4.2,4.,3.8,3.6,3.4
,3.2,3.]
do i=1,#rvec
rRb2=rvec(i)
geometry={Rb;Rb,Rb,rRb2}
basis={
ecp,rb,ecp36sdf;

s,rb,ecp36sdf;
s,rb,1.883135,0.007182,0.003386;
```

rysunek 1. Fragment pliku tekstowego z danymi wejściowymi dla programu MOLPRO

W przypadku programu JMATRIX (Rys. 2), zadanie wprowadzania danych jest nieco ułatwione poprzez opisowe nazwy zmiennych umieszczonych w pliku oraz komentarze.

```
; PROPERTIES OF THE PROJECTILE
e        = 3.0      ; Energy of the projectile
l        = 1        ; Orbital angular quantum number
kappa    = 1        ; Relativistic quantum number
energies = .t.   ; Calculate phase shifts for many energies at once
estart   = 0.1      ; Initial projectile energy for mass calculations
eend     = 10.0      ; Final energy
estep    = 0.1      ; Energy step size

; TYPE OF CALCULATIONS
lambda   = 0.2              ; Scaling parameter
basis    = oscillator        ; Basis set (gauss, laguerre)
scheme   = non-relativistic ; Scheme (relativistic, non-relativistic)
v_light  = finite           ; Velocity of light (finite, infinite)
nstart   = 1                ; Initial value of N
nend     = 2000              ; Final value of N
napprox  = 200

; SCATTERING POTENTIAL
pot_type = well ; Potential type (well, coulomb, yukawa, user)
                ; well: square-well potential, set its parameters below
                ; coulomb: Coulomb-like potential, set its parameters below
                ; yukawa: Yukawa potential, set its parameters below
                ; user: user-specified potential, set it in user potential file
```

rysunek 2. Fragment pliku tekstowego z danymi wejściowymi dla programu JMATRIX

Bardziej zaawansowany interfejs użytkownika opiera się na systemie pytanie-odpowiedź, tak jak w programie GRASP92 (Parpia *et al*, 1995), będącym następcą programu GRASP2. Program składa się z szeregu modułów, zbierających wymagane do swojego działania dane poprzez zadawanie użytkownikowi serii pytań (przykład na Rys. 3 dla modułu GENCSL).

```
 File  grasp92.csl  will be created as the
  GRASP92 Configuration Symmetry List File;
  enter another filename if this is not
  acceptable; null otherwise:

 'Nonrelativistic' scheme?
y
 Enter the list of nonrelativistic core
  subshells:
1* 2*
 Enter the list of nonrelativistic peel
  subshells:
3*
 Redefine default maximum nonrelativistic
  peel subshell occupations?
n
 Redefine default minimum nonrelativistic
  peel subshell occupations?
n
 Enter a nonrelativistic reference peel
  subshell configuration (null if done):
```

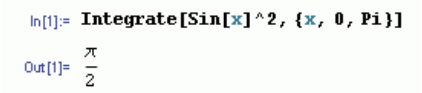
rysunek 3. Fragment interakcji użytkownika z programem GRASP92

Niestety, interfejs ten, w przypadku pomyłki użytkownika na dowolnym etapie wprowadzania danych, wymaga ponownego, żmudnego powtarzania całej procedury od początku. Receptą na to może być wykorzystanie tzw. przekierowania wejścia, polegającego na zapisaniu w pliku tekstowym sekwencji odpowiedzi oddzielonych znakiem końca linii i przekierowaniu tego pliku do wejścia program. W przykładzie z Rys. 3 byłaby to sekwencja:

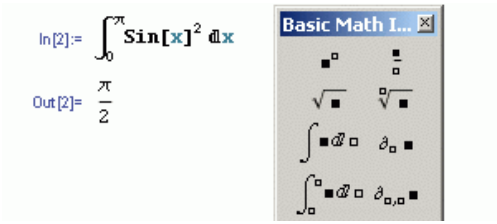
```
y

1* 2*

3*

n

n
```

Powoduje to jednak trudność w ocenie, jakie rzeczywiście odpowiedzi zostały przekazane do programu i utrudnia weryfikację, czy na jakimś etapie nie nastąpiła pomyłka. Przykładowo, pierwszą odpowiedzią w sekwencji pytań jest znak końca linii (oznaczający zaakceptowanie domyślnej nazwy pliku wyjściowego – grasp92.csl), co jednak nie jest widoczne na pierwszy rzut oka w wyżej przedstawionej sekwencji odpowiedzi.

Pakiet obliczeniowy Mathematica posiada zarówno interfejs tekstowy (działający na zasadzie wpisywania poleceń) oraz graficzny (menu oraz wybieranie opcji z okienek dialogowych). Prosty przykład wykorzystania obu typów interfejsu do obliczenia wartości całki przedstawiono na Rys. 4 i 5. Inne programy obliczeniowe tego typu (np. MatLab oraz Maple) działają na podobnej zasadzie.



rysunek 4. Linia komend w programie Mathematica



rysunek 5. Wybór opcji z belki narzędziowej w programie Mathematica

System składu tekstu TeX/LaTeX nie jest co prawda oprogramowaniem stricte obliczeniowym, lecz jest często używany przez naukowców do opracowywania publikacji naukowych. Jest to kolejny, typowy przykład oprogramowania sterowanego z linii komend i przetwarzającego komendy tekstowe. Mimo istnienia na rynku wielu wizualnych edytorów i systemów składu tekstu (np. tych zawartych w popularnych pakietach biurowych), to nic nie wskazuje na to, że TeX/LaTeX zniknie z użycia.

Z kolei program AutoCAD wykorzystywany jest do dwu- i trójwymiarowego komputerowego wspomagania projektowania m.in. przez mechaników czy architektów. Program umożliwia projektowanie poprzez wykreślanie elementów za pomocą myszy, bądź poprzez wpisywanie odpowiednich komend w wierszu poleceń lub przygotowanie odpowiedniego pliku wsadowego przy użyciu języka Lisp, opartego na notacji matematycznej.

Analiza preferencji naukowców, dotyczących interfejsów użytkownika w programach naukowych

Przeprowadziliśmy sondę wśród pracowników naukowych, w której zebraliśmy ich preferencje związane z interfejsem użytkownika w wykorzystywanych przez nich programach, a także opinie dotyczące zasadności budowy GUI dla programów posiadających wyłącznie tekstowy interfejs. Podczas badań zbieraliśmy również informacje dotyczące stylu pracy z programami obliczeniowymi oraz argumenty przemawiające za interfejsem tekstowym i graficznym. Poniżej przedstawiamy wyniki tych badań oraz wypływające z nich wnioski.

Okazało się, że osoby, które używają programów posiadających jedynie tekstowy interfejs, często nie widzą potrzeby tworzenia GUI dla tych programów. Używają tych programów od lat i są przyzwyczajone do tekstowej postaci wprowadzania danych, cenią sobie jej szybkość. Dodatkowo, część programów naukowych dostępnych jest również w postaci kodów źródłowych, a modyfikacja tego kodu (co jest częstą praktyką, np. w celu usprawnienia obliczeń, dodania nowej metody czy algorytmu numerycznego itp.) może wpływać np. na liczbę i rodzaj danych wejściowych/wyjściowych, co z kolei wymagałoby częstych modyfikacji interfejsu. I tak, np. język programowania Fortran jest ciągle powszechnie wykorzystywany m.in. przez fizyków do programowania najróżniejszych obliczeń naukowo-inżynierskich (w szczególności numerycznych). Naukowcy chwalą go ze względu na bogactwo dostępnych bibliotek numerycznych (także do programowania współbieżnego i równoległego) umożliwiających rozwiązanie niemal każdego zadania, oraz szybkość obliczeń. Jednak, aby program napisany

w Fortranie mógł posiadać GUI, musiałby być on napisany w wizualnej wersji języka (Visual Fortran), która jest jednak bardzo mało popularna, bądź też w innym narzędziu i przekazywać parametry obliczeń do rdzenia obliczeniowego napisanego w Fortranie.

O ile okazjonalne zmiany GUI związane np. z dodaniem nowego pola na zmienną są zwykle do zaakceptowania przez naukowców, to zmiany zbyt często następujące i daleko idące (np. związane z wypuszczeniem na rynek kolejnej wersji aplikacji) wywołują zniechęcenie a czasem wręcz irytację. Nie trzeba daleko szukać przykładu, tak było z doskonale znanym większości użytkowników komputera edytorem tekstu MS Word będącym częścią pakietu biurowego MS Office. Zmiana wyglądu GUI pomiędzy poprzednią wersją (*Microsoft Office – Word 2003*) a obecna (*Microsoft Office – Word 2007*) wielu osobom, w tym także i autorom tego artykułu przysporzyła sporo problemów. Nie mówiąc już o tym, ze specyfika pracy niejednokrotnie wymaga korzystania z różnych komputerów gdzie zainstalowane są różne wersje tego samego oprogramowania (zwłaszcza, ze starsza wersja nadal jest popularna). W takiej sytuacji konieczność przypominania sobie: „jak to się robi w tej wersji", powtarza się dość często. Mając tak negatywne doświadczenia z GUI nawet dla programów biurowych, naukowcy wybierają interfejs tekstowy, ponieważ boją się, że zbyt wiele czasu stracą na naukę obsługi nowego interfejsu graficznego, dostarczanego z nową wersją aplikacji.

Z kolei, gdy w programach obliczeniowych jest do dyspozycji zarówno interfejs tekstowy jak i GUI, to ich użytkownicy zaraz po nauczeniu się podstawowych komend potrzebnych do wywoływania najważniejszych funkcji zwykle rezygnują z interfejsu okienkowego na rzecz interfejsów tekstowych (mniej lub bardziej tradycyjnych) lub przetwarzania wsadowego. Głównym powodem takiej decyzji jest wydajność pracy. Przykładowo, o wiele szybciej wprowadza się dane potrzebne do zaprojektowania układów elektrycznych poprzez wprowadzanie kolejnych potencjałów linijka po linijce w polu tekstowym (lub pliku wsadowym), niż poprzez wpisywanie ich w kolejne wyskakujące okienka i akceptowanie ich kliknięciem myszy. I tu pojawia się kolejny problem wskazany przez badanych: mysz, jako urządzenie pośredniczące w interakcji człowiek-komputer w przypadku używania GUI. Wielu z nich uznało konieczność używania myszy jako przeszkodę w płynności pracy (trzeba się oderwać od pisania, aby sięgnąć po mysz i kliknąć w odpowiednie pole). Pojawiły się także opinie mówiące, że konieczność używania myszy przez pewien czas (7-10 dni) wywołała dolegliwości bólowe w stawie łokciowym i barkowym oraz nadgarstku. Fakt występowania dolegliwości spowodowanych używaniem myszy został potwierdzony również w literaturze medycznej (Werner, 2006), problem ten nie może być więc bagatelizowany. Kolejnym argumentem przemawiającym na korzyść interfejsu tekstowego jest możliwość zdalnego uruchomienia obliczeń np. na wydajnym serwerze obliczeniowym za pośrednictwem protokołów Telnet bądź SSH. W ten sposób program (zainstalowany na serwerze) można uruchomić np. z domu bądź dowolnego innego miejsca na świecie, podłączonego do Internetu. Istnieje co prawda możliwość graficznego podłączenia się do zewnętrznego serwera (poprzez tzw. X-klienta), jednak nie jest to tak trywialna sprawa, jak uzyskanie połączenia w trybie tekstowym. Kolejny argument jest już wyłącznie psychologiczny – używając interfejsu tekstowego jest się kimś w rodzaju „guru" – zwykły użytkownik komputera w dzisiejszych czasach

nie potrafi już nawet wywołać linii poleceń DOS, nie wspominając o np. komendach `dir`, `copy` itp., niegdyś będących w powszechnym użyciu.

Interfejs graficzny jest zazwyczaj prostszy, bardziej przejrzysty od tekstowego i pozwala na (w obiegowej opinii) lepsze, bo wizualne kontrolowanie programu. Jednak naukowiec jest człowiekiem z natury podejrzliwym i bardziej ufa sobie, niż programiście – autorowi GUI. Dlatego też z ograniczonym zaufaniem podchodzi do innego niż standardowego, tekstowego interfejsu. „Bo nigdy nie jesteśmy pewni, czy GUI nie psuje naszych obliczeń" – powiedział wprost jeden z naszych rozmówców. Co więcej, niektórzy rozmówcy sugerowali też, że dane wprowadzone do programu bez pośrednictwa GUI mogą być szybciej przetwarzane, przez co wyniki mogą być szybciej dostępne. Obawy te wydawać się mogą irracjonalne, jednak muszą być brane pod uwagę podczas projektowania GUI.

W przypadku systemu TeX/LaTeX, głównym argumentem przemawiającym za jego używaniem mimo istnienia narzędzi wizualnych, jest profesjonalny (i przewidywalny) wygląd dokumentu wynikowego (dużo lepszy niż otrzymany np. z MS Word, szczególnie, gdy treść zawiera dużo wzorów matematycznych). Tekstowa postać pliku źródłowego ułatwia również wymianę plików i wspólną edycję (np. nie występuje problem różnego kodowania znaków narodowych – są one bowiem zapisywane za pomocą podstawowych znaków ASCII poprzedzonych odpowiednią komendą sterującą). Pozostałe argumenty pokrywają się z tymi wymienionymi wyżej. Szczególną zaletą LaTeX-a jest szybkość wprowadzania wzorów matematycznych, pod warunkiem znajomości odpowiednich komend. W przypadku zapomnienia danej komendy, z pomocą przychodzą narzędzia wizualne (wyposażone w menu, belki narzędziowe z symbolami matematycznymi itp. udogodnienia), ułatwiające pracę z systemem. Są to m.in programy Winshell, WinEdt i inne. Umożliwiają one mniej zaawansowanym użytkownikom skorzystanie z menu w przypadku, gdy ci nie pamiętają polecenia koniecznego do wpisania konkretnego elementu np. symbolu czy nawiasu. Czasem jednak możliwości narzędzi wizualnych nie są wykorzystywane – jeden z użytkowników graficznego edytora LaTeX-a np. przyznał, że korzysta co prawda z menu użytkownika, ale używa tylko jednej jego funkcji – przycisku [F5], który pozwala skompilować efekty jego pracy. Tylko nieliczni z spośród zaawansowanych użytkowników przyznają, że używają graficznego edytora LaTeX-a, ponieważ to usprawnia ich pracę.

Nawet rozmówcy, którzy początkowo byli przekonani, że pracują w trybie graficznym, po dłuższej rozmowie przyznawali, że większość zadań (także projektowych) wykonują właśnie poprzez wpisywanie potrzebnych danych w formie tekstu. Mimo naszych początkowych przypuszczeń, że tryb tekstowy wybierają raczej naukowcy-teoretycy, okazało się, że równie chętnie sięgają po niego naukowcy-doświadczalnicy.

Mimo, że interfejs tekstowy uznawany jest za amatorski, jego użytkownicy przyznają, że jest on zwykle dopracowany, sprawdzony i pozwala na realizacje wszystkich funkcji systemu. To wszystko sprawia, że brak (dobrego) GUI dla wielu programów naukowych, obliczeniowych i inżynierskich nie jest tak odczuwalny. Jeden z rozmówców potwierdził, że ewentualny interfejs graficzny dla jego programu najbardziej przydałby się nie jemu osobiście, ale studentom i innym osobom, które z nim współpracują,

a które nie mają czasu na dokładne zapoznawanie się np. ze składnią dostępnych poleceń.

Nieco inaczej ma się sprawa w przypadku osób zajmujących się projektowaniem i obsługą bardziej zaawansowanych zadań (wymagających wizualizacji wyników bądź projektowania/budowania systemu z gotowych elementów), czy też obsługą skomplikowanych urządzeń (lasery, kontrolery przepływu) gdzie niezwykle ważna jest precyzja (np. dokładne określenie temperatury czy innej wielkości fizycznej), a także projektowaniem stron internetowych (choć rzecz jasna nie jest to już zadanie bezpośrednio związane z obliczeniami naukowo-inżynierskimi). W takich przypadkach możliwość wizualnego podglądu wyników czy też wprowadzania danych jest niewątpliwą zaletą, umożliwiającą szybkie reagowanie na informacje otrzymywane z urządzenia. Uwagi dotyczące wizualnego projektowania stron WWW nieco zaskoczyły autorów artykułu, którzy raczej preferują standardowe metody tworzenia układu strony (czyli starannie odmierzenie pikseli i wpisanie ich do plików CSS – kaskadowych arkuszy styli) od budowy aplikacji webowych za pomocą specjalnych edytorów 'z klocków', co staje się jednak coraz bardziej popularne.

Innymi zaletami interfejsów graficznych w oczach naukowców są: możliwość dotarcia z programem do szerszego grona (a także młodszych) użytkowników, możliwość uniknięcia żmudnej czynności ponownego wpisywania wszystkich danych np. w przypadku pomyłki polegającej na podaniu niewłaściwej wartości jakiegoś parametru w przypadku systemu pytanie-odpowiedź oraz szybkość nauki programu.

Proces inżynierii wymagań dla definiowania GUI dla aplikacji naukowych oraz wybrane realizacje

Po analizie preferencji i trybu pracy naukowców z programami obliczeniowymi, staraliśmy się znaleźć odpowiedzi na pytania, czy zalety GUI (w kontekście tego specyficznego typu oprogramowania) mogłyby skłonić naukowców do zaakceptowania interfejsów graficznych oraz co ewentualnie skłoniłoby ich do całkowitej rezygnacji z tekstowego interfejsu na rzecz graficznego? W wyniku przeprowadzonej analizy sformułowaliśmy warunki, które powinny zostać spełnione, aby to mogło nastąpić, a które przedstawimy w dalszej części rozdziału.

W swojej praktyce i najbliższym otoczeniu spotkaliśmy się z kilkoma próbami stworzenia interfejsu graficznego dla istniejącego programu obliczeniowego. Jedna z nich zakończyła się porażką i odłożeniem stworzenia GUI satysfakcjonującego przyszłego użytkownika na bliżej nieokreśloną przyszłość (dwie pozostałe próby zakończyły się częściowym powodzeniem i zostały opisane w dalszej części rozdziału). Pierwotnie wskazywano na czasochłonność projektu jak główny powód niepowodzenia. Dłuższa rozmowa ujawniła, że główna przyczyna problem leży gdzie indziej. Otóż, okazało się, że do wykonania interfejsu graficznego dla aplikacji naukowej, obliczeniowej bądź inżynierskiej nie wystarczy umiejętność tworzenia interfejsów użytkownika w ogóle. Projektant takiego specyficznego interfejsu powinien także posiadać przynajmniej podstawowe informacje na temat dziedziny problemu. Jest to bardzo ważny wniosek, który musi być brany pod uwagę w inżynierii wymagań przy definiowaniu GUI dla programów naukowych. Niestety, jest to wymóg dość trudny w praktycznej realizacji, ze względu na to,

że większość naukowców nie jest na tyle dobrymi projektantami i programistami, żeby samodzielnie stworzyć GUI, z kolei informatycy – projektanci i zawodowi programiści nie posiadają zwykle odpowiedniej wiedzy dziedzinowej. Próba komunikacji pomiędzy tymi dwoma grupami często z góry skazana jest na niepowodzenie. Na przykład, naukowiec nie chce tracić zbyt wiele czasu na rozmowy z projektantem GUI, bo np. ma do wykonania pilne obliczenia (wskutek nieustannej presji na osiąganie wyników), które zawsze może uruchomić poprzez interfejs tekstowy, w konsekwencji nowo powstałe GUI nie spełnia odpowiednich jego wymagań – bo po prostu nie zostały odpowiednio sformułowane i przekazane. Takie GUI z kolei nie satysfakcjonuje naukowca, który w ten sposób utwierdza się w przekonaniu, że interfejs tekstowy jest optymalnym wyborem.

Niedostateczna jakość interfejsów graficznych w programach naukowych, obliczeniowych i inżynierskich spowodowana jest tym, że są to często aplikacje niskobudżetowe (wiele z programów dedykowanych używanych jest przez wąską grupę specjalistów) bądź wręcz tworzone przez samych naukowców jedynie na własne potrzeby. Na zaprojektowanie dobrego (użytecznego dla większej grupy osób) interfejsu zwyczajnie nie wystarcza czasu ani środków finansowych. Ponadto naukowiec, który tworzy program na własne potrzeby prawdopodobnie nawet nie zdaje sobie sprawy z tego, ze jego aplikacja może okazać się zupełnie nieużyteczna dla innych osób. Kończy się to zwykle tym, że nawet jeżeli stworzony zostanie GUI dla danego programu, to np. menu użytkownika jest bardzo ubogie, brak w nimi wielu, nawet podstawowych funkcji, skrótów klawiaturowych itp. A przecież udostępnienie własnej, łatwej w użyciu aplikacji innym osobom mogłoby skutkować znaczącym wzrostem cytowań jej autora, co jest nawet bardziej pożądane od liczby publikacji. Podobne doświadczenia opisane zostały w literaturze (Spolsky, 2001). Na umieszczonym tam przykładzie (doświadczenia Dave Winera) widać, jak ważny jest kontakt z potencjalnym klientem tworzonego oprogramowania.

Na podstawie analiz oczekiwań użytkowników oprogramowania naukowego, sformułowaliśmy zbiór ogólnych wymagań, którymi kierować powinni się projektanci GUI. Przede wszystkim należy:

- stosować:

  - standardy de facto dotyczące np. użycia zwyczajowych skrótów klawiaturowych;

  - w większym zakresie menu kontekstowe (np. dodanie większej liczby funkcji do menu pomocniczego, wywoływanego prawym przyciskiem myszy);

  - klasyczny, estetyczny i skromny wygląd GUI, bez zbędnych dodatków. GUI powinien być intuicyjny i ergonomiczny, czyli taki, którego obsługi nauczyć można się nawet bez odwoływania się do pomocy zawartej w programie czy korzystania z podręczników, tutoriali, natomiast dostęp do udostępnianych funkcji powinien być logiczny i uporządkowany;

  - kontrolę wprowadzanych danych (interfejs powinien sprawdzać, czy spełnione są zależności między wprowadzanymi parametrami, albo czy np. podana została

liczba całkowita a nie zmiennoprzecinkowa itp.);

- wieloplatformowość (interfejs powinien działać pod różnymi systemami sprzętowymi i programowymi – co najmniej Windows i Linux; wskazany Mac);

- pozwolić użytkownikowi na:

  - wywoływanie poprzez GUI rdzenia obliczeniowego umieszczonego na zdalnym serwerze (odpowiednik logowania poprzez Telnet / SSH);

  - podgląd wyników na bieżąco;

  - wizualizację wyników (np. rysowanie na bieżąco wykresów);

  - samodzielną konfigurację i personalizację programu. Postulowano wręcz o zastosowanie totalnego minimalizmu konfiguracyjnego przy pierwszym uruchomieniu programu, tak by użytkownik mógł sam go skonfigurować w zależności od swoich potrzeb;

- pamiętać, że:

  - mimo wszystko, zawsze pożądana jest wersja tekstowa interfejsu, na wypadek konieczności uruchomienia obliczeń w niesprzyjających okolicznościach (np. dostęp wyłącznie do konsoli tekstowej);

  - mile widziana byłaby dostępność kodu źródłowego interfejsu bądź jego wysoka elastyczność (gdy zmieni się kod źródłowy aplikacji, powinna istnieć możliwość łatwej modyfikacji interfejsu, np. polegającej na dodaniu nowego pola tekstowego na nowo dodaną zmienną, dodaniu nowej pozycji menu itp.);

  - użytkownik musi mieć pewność, że GUI nie wprowadzi niepożądanych modyfikacji we wprowadzanych danych, np. nie zmieni typu zmiennej;

  - pamięć jest chwilowa. Użytkownik pamięta komendy tylko wtedy, gdy ich używa. Zawsze powinien mieć pod ręką dobry podręcznik oraz użyteczny, interaktywny system pomocy.

Chcielibyśmy trochę uwagi poświęcić ostatniemu postulatowi, dotyczącemu systemowi pomocy. Wielu naszych rozmówców wyraziło potrzebę wzbogacenia programów, których używają do pracy o interaktywną (inteligentną) pomoc/podpowiedź. Nawet ci, którzy sami nie wyrazili takiej potrzeby, po usłyszeniu pytania o nią wyrazili zainteresowanie. Jakie zadanie miałaby realizować ta funkcjonalność? Chodzi głównie o zastosowanie systemu, np. podpowiadania składni, odpowiedniej kolejności działań itp. informacji. Wzbogacenie dotychczasowych programów o te funkcje w wielu przypadkach w zupełności wystarczyłoby użytkownikom, przez co budowanie skomplikowanego i zaawansowanego menu użytkownika stałoby się niepotrzebne. Jednym z najlepszych przykładów dobrze zaprojektowanego interaktywnego systemu pomocy jest ten wbudowany w GUI dla języka Python (system

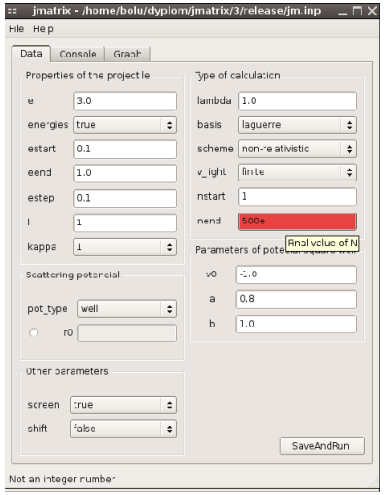
IDLE), dość powszechnie stosowanego przez naukowców do krótkich, pomocniczych obliczeń. Szczegółowa analiza tego zagadnienia wykracza jednak poza tematykę niniejszego opracowania, zainteresowanych odsyłamy więc np. do publikacji internetowych, np. do strony http://zetcode.com/tutorials/pythontutorial/interactivepython/.

Jak wspomniano wcześniej, kolejne próby stworzenia GUI, z którymi się spotkaliśmy, zakończyły się częściowym sukcesem (Duma, 2007). Polegały one na stworzeniu GUI dla programów JMATRIX (jego interfejs tekstowy został przedstawiony na Rys. 2) oraz GRASP92 (Rys. 3). Spośród istniejących kilku narzędzi i bibliotek mogących posłużyć do wykonania zadań (MFC, Qt, GTK+, Kylix) wybrano Qt, ze względu na jego wieloplatformowość, dużą elastyczność, obiektowość, łatwość kompilacji oraz użyteczne narzędzia pomocnicze. Przygotowaniem GUI zajęła się osoba (dyplomant kierunku Informatyka Stosowana, WFTiMS, PG) posiadająca niezbędną wiedzę informatyczną, a przy tym podstawy dziedzinowe. Nie dysponowała ona tak szeroką analizą wymagań jak bieżąca, jednak częsty kontakt ze zleceniodawcą (promotor pracy dyplomowej – współautor niniejszego artykułu) sprawił, że część wymagań określonych w tym rozdziale została spełniona.

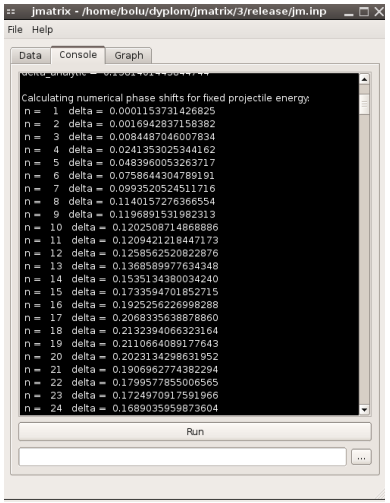
Wygląd GUI dla programu JMATRIX został przedstawiony na Rys. 6-8. Okno programu składa się z trzech części (zakładek), pierwsza z nich odpowiada za wprowadzanie danych wejściowych, wraz z kontrolą ich poprawności (Rys. 6; na czerwono podświetlona nieprawidłowa wartość; na dole, w oknie statusu znajduje się opis błędu). Po wpisaniu danych oraz wciśnięciu przycisku „Save & Run” zapisywany jest na dysku plik tekstowy, analogiczny do tego z Rys. 2 oraz uruchamiany jest automatycznie właściwy program, który z kolei wczytuje wygenerowany plik oraz odpowiada za przeprowadzenie właściwych obliczeń. Druga zakładka (Rys. 7) zawiera tekstowy podgląd obliczeń (z przyciskiem przerwania / kontynuacji / ponownego uruchomienia obliczeń), trzecia zaś wizualizuje obliczenia poprzez rysowanie poglądowego wykresu na bieżąco, podczas trwania obliczeń (Rys. 8). Interfejs jest wieloplatformowy, jego wygląd na wiodących platformach sprzętowych i programowych jest identyczny. GUI spełnia też (wg. opinii autorów artykułu) wymóg dotyczący estetyki i ergonomii.

Należy podkreślić, że stworzenie GUI nie wymagało żadnej ingerencji w kod właściwego programu (napisanego w Fortranie). Interfejs użytkownika jest w tym przypadku samodzielnym bytem – użytkownikowi pozostawiona została możliwość używania interfejsu tekstowego. Dostępny jest również kod źródłowy interfejsu, przez co możliwa jest jego modyfikacja w przyszłości, gdy np. zmieni się format pliku wejściowego.

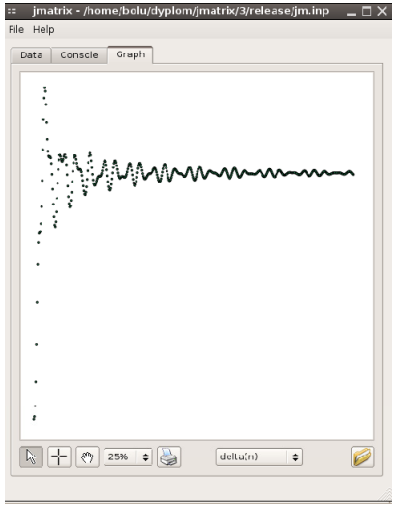
Znaczącym ograniczeniem stworzonego GUI jest brak możliwości wywoływania obliczeń na zdalnym serwerze. Nie ma żadnych technicznych przeszkód w zaimplementowaniu takiej funkcjonalności, jest to jednak sprawa dość trudna i przekroczyła możliwości dyplomanta.



rysunek 6. JMATRIX-GUI. Okno wprowadzania danych (źródło: Duma, 2007)

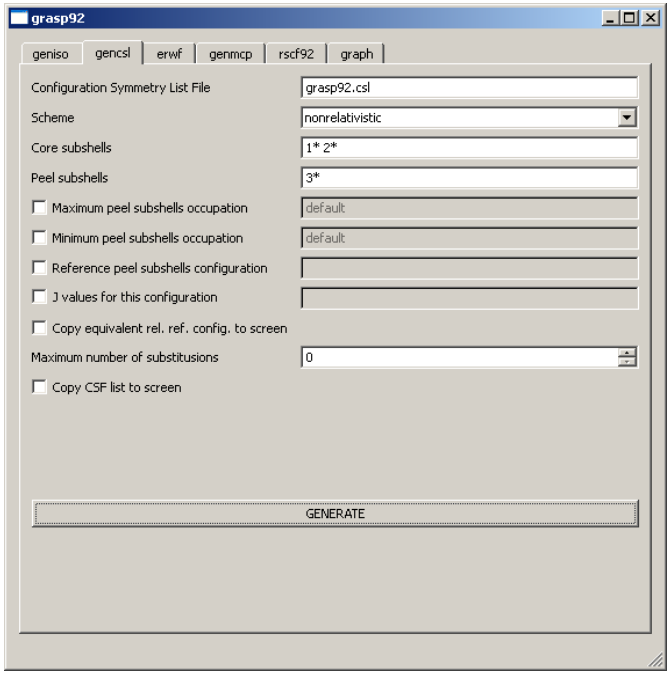


rysunek 7. JMATRIX – GUI. Podgląd wyników (źródło: Duma, 2007)



rysunek 8. JMATRIX-GUI. Wykres poglądowy, rysowany na bieżąco (źródło: Duma, 2007)

Stworzenie GUI dla programu GRASP92 było zadaniem nieco odmiennego typu, ze względu na zastosowany oryginalnie system pytanie-odpowiedź (Rys. 3). Stworzony GUI (Rys. 9) generuje więc nie plik konfiguracyjny, lecz sekwencję odpowiedzi, tak jak to zostało przedstawione wcześniej w opisie programu GRASP92.



rysunek 9. GRASP92-GUI. Okno wprowadzania danych dla modułu GENCSL

W powyższym przykładzie, po wciśnięciu przycisku „GENERATE" został utworzony plik o zawartości:

```
grasp92.csl
y
1* 2*
3*
n
n
n
0
n
```

który następnie jest już bezpośrednio przekierowywany do właściwego modułu (w tym wypadku GENCSL). Okno programu składa się z szeregu zakładek, odpowiadających poszczególnym modułom oraz ostatniej, zawierającej wizualizację wyników. Użytkownik może prześledzić poprawność działania programu poprzez podgląd zawarty w konsoli tekstowej, uruchamianej automatycznie po wciśnięciu „GENERATE", tym razem jednak w osobnym okienku.

Jak można zauważyć z powyższego opisu i porównania rysunków, GUI dla programu GRASP92 jest bardzo podobny do tego stworzonego dla programu JMATRIX, zaimplementowano podobne funkcjonalności (w tym konsolę oraz rysowanie wykresów). Prawdziwe zatem pozostają i w tym przypadku sformułowane wyżej, dla JMATRIX-GUI, jego zalety oraz ograniczenia. Jak widać, obie te realizacje spełniają więc znaczną część postulowanych wymagań. Jednak bardzo interesujący jest fakt, że używane są jedynie okazjonalnie, szczególnie w celu zademonstrowania oprogramowania innym naukowcom, w celu zachęcenia ich do jego używania. Zasadnicza część obliczeń jest jednak ciągle realizowana przy użyciu wyłącznie tekstowego interfejsu. Stąd wspomniano wyżej o jedynie częściowym sukcesie przedsięwzięcia polegającego na stworzeniu GUI dla programów JMATRIX i GRASP92. Wydaje się, że decydujące znaczenie w jedynie częściowym powodzeniu przedsięwzięcia mógł mieć wspomniany brak możliwości wywoływania rdzenia obliczeniowego na zdalnym serwerze. Dodanie takiej funkcjonalności znacząco podniosłoby wartość GUI, szczególnie w przypadku konieczności prowadzenia długotrwałych obliczeń – umożliwiłoby bowiem wykonanie ich nie na komputerze lokalnym, lecz na

zdalnym serwerze, dedykowanym do przeprowadzania obliczeń.

Podsumowanie

Celem niniejszego opracowania było przybliżenie czytelnikowi zagadnień związanych z projektowaniem i tworzeniem interfejsów użytkownika dla programów naukowych, obliczeniowych oraz inżynierskich. Przedstawiliśmy przykładowe interfejsy tekstowe oraz graficzne, a także zbadaliśmy oraz zanalizowaliśmy preferencje naukowców dotyczące obu typów interfejsu. Okazało się, że interfejs tekstowy jest jeszcze bardzo powszechnie stosowany, dodatkowo wielu naukowców nie widzi potrzeby zmiany tej sytuacji. W wyniku przeprowadzonych analiz sformułowaliśmy szereg ogólnych wymagań, które powinny być spełnione, żeby interfejs graficzny mógł zastąpić interfejs tekstowy. Przedstawiliśmy przykładowe realizacje GUI dla istniejących programów naukowych.

Otwarte pozostaje pytanie, czy firmy produkujące komercyjne oprogramowanie naukowe mogłyby osiągnąć wymierne korzyści biznesowe poprzez dodanie graficznych interfejsów użytkownika do programów komercyjnych (np. MOLPRO). Wydaje nam się, że mogłoby się tak stać pod warunkiem spełnienia niemal wszystkich wymagań, przedstawionych w niniejszym opracowaniu. Środowisko naukowców okazało się bowiem bardzo konserwatywne jeżeli chodzi o preferencje dotyczące interfejsu użytkownika. Wydaje się, że dopiero spełnienie znakomitej większości przedstawionych wymagań mogłoby przynajmniej część z nich skłonić do zrezygnowania z interfejsów tekstowych na rzecz graficznych. Jest to jednak dość trudne ze względu na specyficzny charakter aplikacji naukowych.

Czy tradycja projektowania Kansei mogłoby znaleźć zastosowanie w przypadku definiowania GUI dla aplikacji naukowych? Wydaje się, że tak, lecz jednak w bardzo ograniczonym zakresie. Pozostawiamy ten problem otwarty dla kolejnych badań dotykających poruszonej przez nas tematyki aplikacji naukowych, obliczeniowych i inżynierskich.

Literatura

[1] J. Spolsky; „Projektowanie interfejsu użytkownika. Poradnik dla programistów”; Mikom; 2001
[2] J. Johnson; „*GUI Bloopers*: Don'ts and Do's for Software Developers and Web Designers”; Academic Press; 2000
[3] C. Akass; „The men who really invented the GUI”; Personal Computer World 04; 2001
[4] P. Syty; „The J-matrix method: Numerical computations”; TASK Quarterly 3 No. 3, 269; 1999
[5] F.A. Parpia, C. Froese Fischer, I. P. Grant; „GRASP92: a package for large-scale relativistic atomic structure calculations”; Comput. Phys. Commun. 94, 249; 1996
[6] M. Duma; „Tworzenie graficznych interfejsów użytkownika dla programów obliczeniowych” – praca dyplomowa; Politechnika Gdańska, Wydział Fizyki Technicznej i Matematyki Stosowanej; 2007
[7] R. Werner; „Evaluation of work-related carpal tunnel syndrome”; J Occup Rehabil. 16 (2); 2006