INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

Inwestujemy w Waszą przyszłość

INSTEPRO
Zintegrowane Sterowanie Produkcją

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ ROZWOJU REGIONALNEGO

Raport wewnętrzny projektu InStePro

Nr 4.4: Projekt niestandardowych algorytmów regulacyjnych

Data

30.09.2010

Przygotował:

J. Augustyn

Raport zawiera 3 części.

Część #1:
Projekt zakresu, konsultacje oraz pilotowanie przedprototypowej realizacji centralnego modułu INSTEPRO w środowisku QNX ver. 6 do współpracy z niestandardowymi algorytmami regulacyjnymi

Część #2:
Możliwości prezentacji działania modułów INSTEPRO w środowisku Windows.

Część #3:
Temat: Opracowanie i przeprowadzenie testów modułu wielowymiarowych modeli liniowych dla potrzeb regulatora predykcyjnego.
Obszar:
Realizacja zadania 4.4. Projekt niestandardowych algorytmów regulacyjnych.
Realizacja modułów systemu (w zakresie obejmującym treść zadania 4.4 – niestandardowe algorytmy regulacji), obszar zadania 5.2.
Projekt i przygotowanie zestawu testów funkcjonalnych i testowanie modułów (w zakresie obejmującym treść zadania 4.4 – niestandardowe algorytmy regulacji), obszar zadania 5.2.
Realizacja zadania 'Projekt interfejsów' (w zakresie 4.4 – niestandardowe algorytmy regulacji).

1. W ramach: Część #1.

Zastosowanie systemu operacyjnego czasu rzeczywistego QNX w wersji v.6.4.1 (który był najnowszą dostępną w chwili zakupu wersją tego systemu) przesuwa technologicznie rozwiązanie o 15 lat do przodu w stosunku do wersji 4. Stosowanie dostępnych w nim narzędzi może znacząco ułatwić oraz przyspieszyć realizację poszczególnych zadań czy też realizacji fragmentów projektu INSTEPRO.

Możliwość raportowania, że system sterowania lub jego niektóre moduły są realizowane obiektowo, jest zdaniem autora raportu istotne z punktu widzenia nowoczesności rozwiązania i wymagań stawianych projektom Unijnym z obszaru Programy Operacyjne Innowacyjna Gospodarka.

Autor raportu podjął się przekazania kluczowych idei zastosowanych w module centralnym Instepro, rozkodowania i interpretacji niektórych nazw własnych, przygotowania projektu zakresu migracji, wykonywał niezbędne bieżące konsultacje oraz pilotował realizację wspólnie z p. Jackiem Szmydem.

Została zaimplementowana przedprototypowa realizacja modułu centralnego. Podstawą realizacji początkowo była wersja projektu numer 86, a następnie najbardziej aktualna w chwili realizacji wersja 186.

Została zachowana możliwość pracy pod systemem QNX v.4. Decyzja o migracji została przekazana do dalszych decyzji kierownictwu projektu.

Autor raportu dziękuje panu J.T.Dudzie, a także M. Klemiato, M.Pełechowi za możliwość realizacji tej części projektu i nabyte doświadczenia. Dodatkowo dziękuje także panu W.Gredze i M.Nowakowi, którzy w mniejszym, lecz także znaczącym stopniu mieli wpływ na realizację i doświadczenia. Autor wyraża także wdzięczność panu Jackowi Szmydowi.

2. W ramach: Część #2.

Możliwości prezentacji działania modułów INSTEPRO w środowisku Windows.

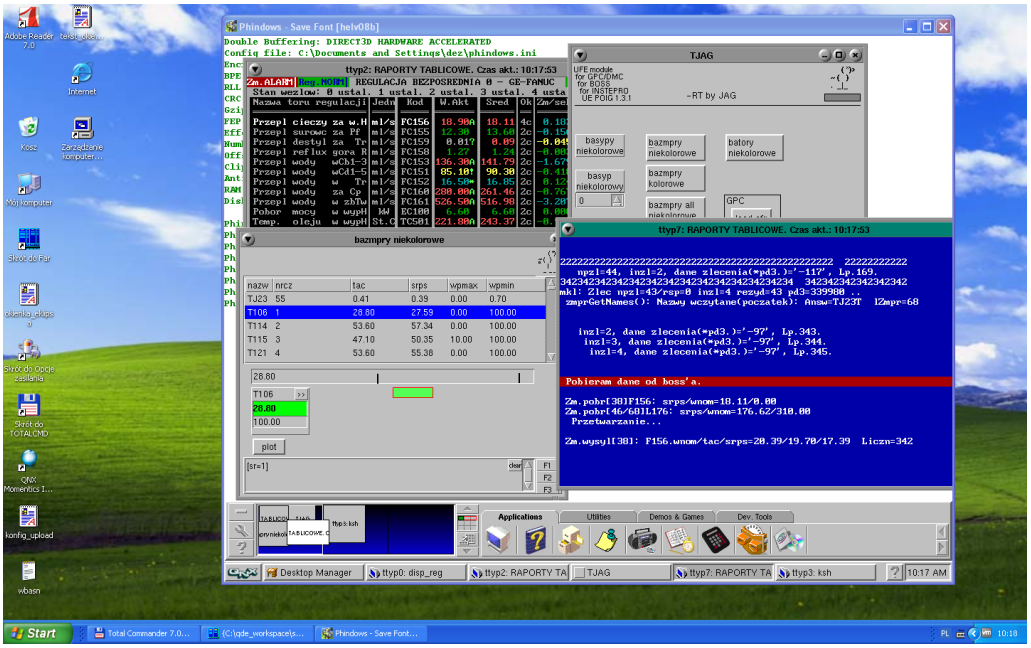
Wygląd ekranu Phindows z działającym Instepro pod systemem v4 pokazano na rys 2.1.1-2.

Grafika wektorowa - Przekazywane komunikatami wysoko poziomowymi [lit: dokumentacja phindows].

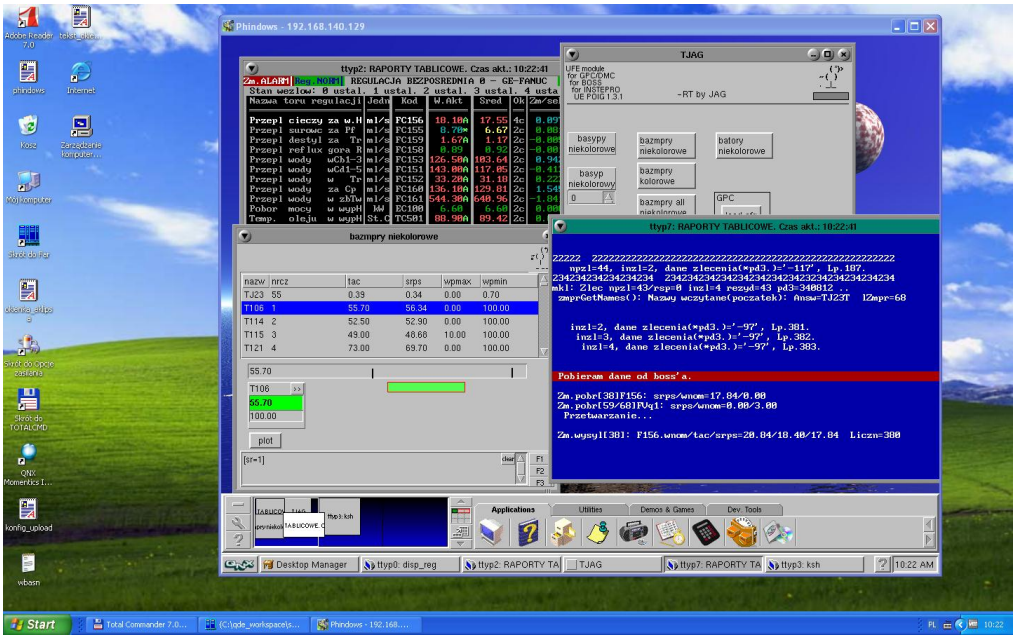
Windows używa DirectX/OpenGL [lit: dokumentacja phindows], zatem wpływ na szybkość ma używana karta graficzna w środowisku Windows oraz jej sterowniki.

Zalety rozwiązania: nie jest wymagane tworzenie komunikatów, ramek, itd., oraz nie trzeba pisać programów QNX->TCP, TCP->WIN, WIN->GUI. Główny zysk polega zatem na eliminacji strat czasu podczas tworzenia aplikacji.

Interakcje pomiędzy systemami: może wystąpić „spowolnienie pracy” przy kreśleniu zależnie od jakości łącza, głównie przy wyświetlaniu bitmap.



Rys. 2.1.1. Phindows / QNX v4



Rys. 2.1.2. Phindows / QNX v4

3. W ramach: Część #3.

Temat: Opracowanie i przeprowadzenie testów modułu wielowymiarowych modeli liniowych dla potrzeb regulatora predykcyjnego.

Realizacja zadania 4.4. Projekt niestandardowych algorytmów regulacyjnych.

Realizacja modułów systemu (w zakresie obejmującym treść zadania 4.4 – niestandardowe algorytmy regulacji), obszar zadania 5.2.

Projekt i przygotowanie zestawu testów funkcjonalnych i testowanie modułów (w zakresie obejmującym treść zadania 4.4 – niestandardowe algorytmy regulacji), obszar zadania 5.2.

Zadanie obejmowało:
- Opracowanie projektu jądra wielowymiarowego modelu liniowego.
- Opracowanie projektu integracji z modułem regulatora predykcyjnego.
- Opracowanie narzędzi wspomagających implementację i testowanie modułu.

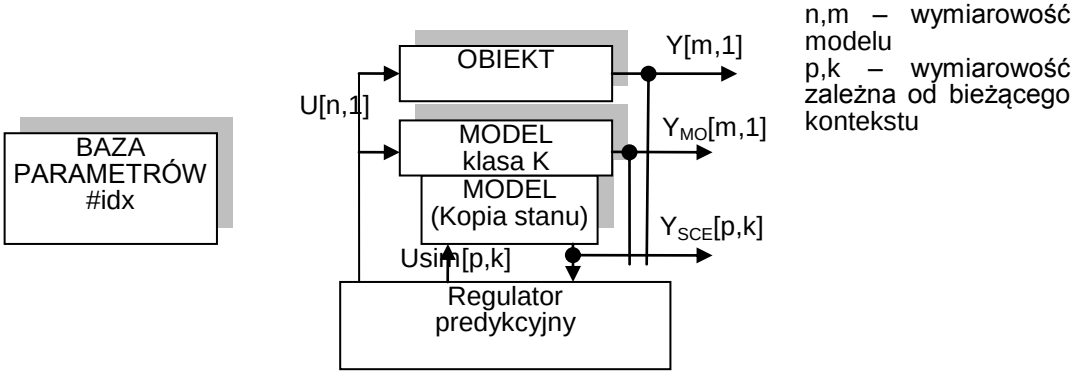
Uwaga: w niniejszym raporcie słowa „model" oraz „obiekt" będą używane w dwóch różnych znaczeniach:
- wielowymiarowego modelu/obiektu liniowego - w odniesieniu do modelu/obiektu sterowania,
- modelu/obiektu w znaczeniu informatycznym.

Znaczenie powinno być rozróżniane przez kontekst wypowiedzi.

3.1. Podstawy (background)

Zgodnie z powszechnie przyjętą podczas realizacji projektów informatycznych zasadą inkapsulacji [lit: programowanie obiektowe], zostały rozdzielone funkcje modelu od jego funkcji interfejsowych oraz funkcje regulatora predykcyjnego od jego funkcji interfejsowych. Taki podział ułatwia niezależne rozwijanie poszczególnych części zadania oraz późniejszą integrację modułów.

Przyjęty model ogólny przedstawiono na rys. 3.1. Pokazuje on powiązania między modułami.



Rys. 3.1. Model ogólny powiązań między wielowymiarowym regulatorem a wielowymiarowymi modelami dla konkretnego przypadku (konfiguracji)

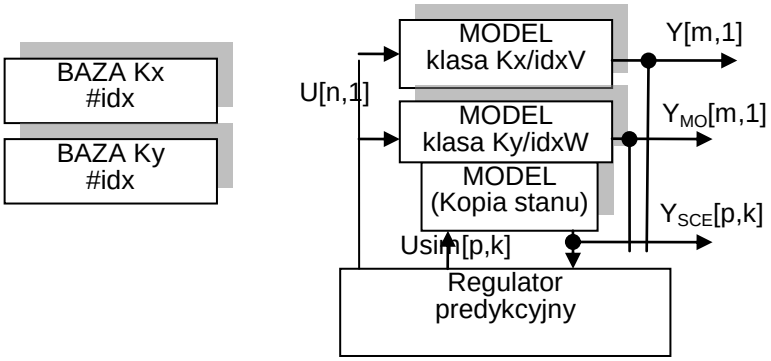
Przedstawiony na tak wysokim poziomie abstrakcji model podsystemu, w praktyce może służyć jedynie ilustracji idei. Nie ma on jednak praktycznego znaczenia na etapie projektowania i implementacji. Wymaga dalszego sprecyzowania, m.in. kto, kiedy i jakie dane będzie przesyłał, ustalenia ich formatów. Będzie to opisane w dalszej części raportu.

Początkowo zostaną przedstawione tzw. wybrane przypadki użycia [lit. Uml].

Case study:

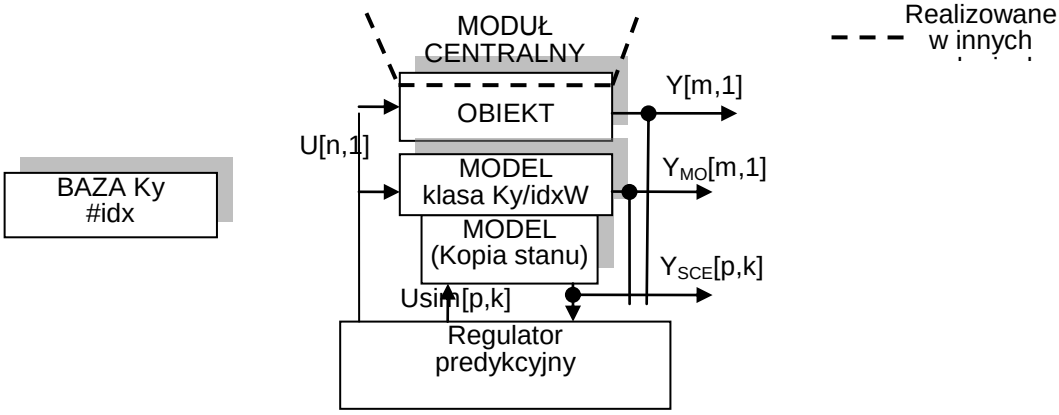
Poniżej opisano trzy wybrane przypadki typowego użycia podsystemu regulator-model-obiekt.

a. Zakłada się, że funkcję rzeczywistego obiektu regulacji zastępuje jego model. Przypadek jest bardzo użyteczny w pierwszym etapie implementacji oraz testowania modułów i przeznaczony do wykonywania głównie symulacji i walidacji wyników. Pozwala na rozwijanie modułu bez konieczności współpracy z innymi częściami systemu INSTEPRO.



Rys. 3.2. Case study: przypadek a.

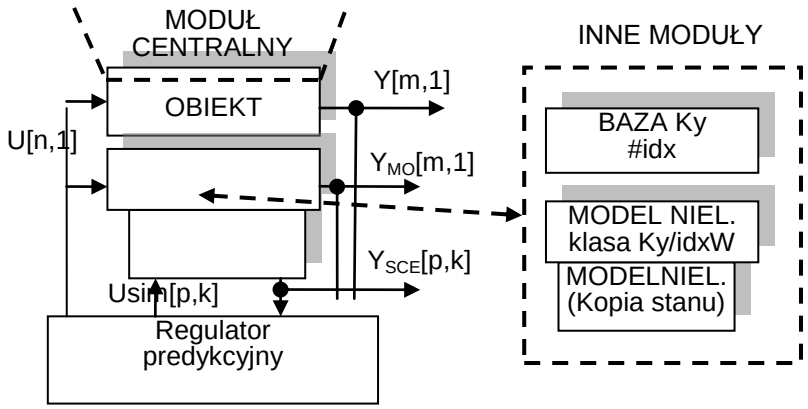
b. Konfiguracja docelowa. W tym przypadku, obiekt jest powiązany z rzeczywistą instalacją poprzez funkcje dostarczane przez centralny moduł.



Rys. 3.3. Case study: przypadek b.

c. Konfiguracja docelowa z modelem nieliniowym.

W tym przypadku, regulator posługuje się modelem nieliniowym dostarczonym przez inne moduły projektu. Sposób przekazywania danych (client-server, wspólna przestrzeń adresowa, itp.) może być zorganizowany dowolnie przez stronę realizującą model nieliniowy, od strony regulatora musi jednak używać opisanego dalej zestawu funkcji.



Rys. 3.4. Case study: przypadek c.

3.2. Założenia projektowe

Podczas projektowania przyjęto następujące wytyczne i założenia (metodologia):

Z1. Na podstawie ogólnej wiedzy dotyczącej algorytmu regulatora predykcyjnego i wymaganych danych składowych, algorytmu wielowymiarowych modeli liniowych oraz ich danych składowych oszacowano złożoność informatyczną modułu regulator-model-obiekt na 20-30 tysięcy linii kodu źródłowego. Znaczna część tego kodu będzie miała charakter numeryczny. Przedstawiany fragment dotyczący wielowymiarowych modeli liniowych dla potrzeb regulatora predykcyjnego szacowany jest na 10 tysięcy linii kodu.

Z2. Przyjęto, że poszczególne części będą projektowane obiektowo (w sensie informatycznym) i będą wykorzystywać zalecania projektowania obiektowego (np. enkaspulacja, dziedziczenie, funkcje wirtualne). Taka metodologia jest zalecana podczas projektowania większych systemów w wielu źródłach [lit: projektowanie obiektowe]. Projekt obiektowy pozwala na czytelniejszy opis struktury systemu.

Z3. Implementacja będzie wykonana w języku obiektowym (C++) z wykorzystaniem istniejących w nim mechanizmów (funkcje wirtualne, dziedziecznie, szablony) [lit: projektowanie/programowanie obiektowe].

Podczas realizacji przyjęto, że komponent realizujący wielowymiarowy model będzie obiektem w rozumieniu programowania obiektowego.

Z4. Projekt modułu modelu powinien mieć charakter uniwersalny w celu zapewnienia możliwości elastycznej implementacji różnych typów (np. nieliniowe).

Z5. W celu ułatwienia późniejszej integracji, zarówno z modułem regulatora predykcyjnego jak i pozostałymi modułami INSTEPRO zostanie opracowany specjalny obiekt pośredniczący.

Jego konstrukcja nie powinna ograniczać ani krępować implementacji od strony regulatora, a z drugiej musi pasować do realizowanych rozwiązań w języku C oraz umożliwiać programistom używającym jedynie tego języka rozszerzanie modułu.

Założenie Z5 wynika z przyjętej w INSTEPRO zasadzie realizacji modułów w języku C. Pozwala jednak na ominięcie specyfiki kompilatora C++, jego wywołania, itd. Pozwala także na łączenie modułow na poziomie półkompilatów („.obj”).

Przyjęte w punktach Z2, Z3 założenia dają podstawy do zraportowania, że moduły projektu INSTEPRO (być może będzie to tylko ten moduł) zostały zrealizowane z użyciem nowoczesnych metod projektowania i programowania. Realizacja przynajmniej jednego takiego wydaje się konieczna w kontekście projektu o tematyce innowacyjna gospodarka.

3.3. Opis projektu

Na wysokim poziomie abstrakcji, zaprojektowano klasę podstawową o nazwie cOB2Generic. Klasę można traktować jako abstrakcyjną. Zawiera ona zestaw usług jakie powinien dostarczać model. Dedykowane i szczegółowe implementacje modeli muszą go zachować. Dotyczy to zarówno opisywanych wielowymiarowych modeli liniowych jak i realizowanych w ramach innych części projektu wielowymiarowych modeli w tym modeli nieliniowych.

W modelu abstrakcyjnym zaprojektowano taki zestaw usług, aby konieczność ingerencji w kod regulatora przy rozszerzaniu jego możliwości była możliwie jak najmniejsza. Wymagane jest bowiem, aby regulator predykcyjny posiadał szerokie możliwości konfiguracji jak też i obliczania (symulowania) wariantowych scenariuszy zachowania modelu sterowania np. przy różnych jego parametrach.

Zapewnienie współpracy ze innymi (nieobiektowymi) częściami projektu INSTEPRO będzie odbywało się za pomocą szczegółowych realizacji klas pochodnych od cOB2Generic. Od strony innych modułów INSTERPO będą reprezentowane jako zestaw zwykłych funkcji w języku C. Pozostali realizatorzy będą mogli zatem wykorzystywać jedynie zwykłe techniki języka C bez konieczności zaznajamiania się ze specyfiką klas, obiektów czy kompilatora C++. Od strony związanej z regulatorem predykcyjnym, obiekty pochodne dalej będą podlegały wymienionym wcześniej regułom, co pozwala na wykorzystanie całej gamy korzyści wynikającej z np. możliwości dziedziczenia, funkcji wirtualnych, czy nawet łączenia modułów na etapie linkowania (bez wymagania posiadania kodu źródłowego).

Poniżej przedstawiono projekt klasy abstrakcyjnej. Precyzuje on opis współpracy między modułami regulatora i modelu, który został przedstawiony w postaci „strzałeczek” na rys. 3.1.

cOB2Generic

```
//-------------------------------------------------------------------------
class cOB2Generic{
public:
        cOB2Generic( int _idxCfg, PtWidget_t *_errinfo, PtWidget_t *_info );
         #define  ONLY_GET_CFG  (char)1
        cOB2Generic( char onlyCfg, int _idxCfg, PtWidget_t *_errinfo, PtWidget_t *_info );
        virtual ~cOB2Generic();

        #define cOB2Generic_CLASS_TYPE   0
        #define cMol2_CLASS_TYPE         1
        #define cMoBoss_CLASS_TYPE       2
        #define cOb2Boss_CLASS_TYPE      3
        #define cMoModeller4_CLASS_TYPE  4

        int getCfgGLUT( void );

        void virtual getOBInfo( sMOInfo *_MOInfo );

        int virtual setU( jMtx &U );              // jednoprobkowy, rzeczywisty-wysyla sterowanie, model-wykonuje jeden krok
        int virtual getLastU( jMtx &U ); // jednoprobkowy, pobiera ostatnio wdrozone, wykonuje jeden krok
        int virtual getY( jMtx &Y );              // jednoprobkowy, ostatni pomiar

        int virtual Simulate( jMtx &Uwym, jMtx &Yodp, int idxFE=0 ); // U[Nu,len], Y[Ny,len]
        int virtual SimulateCopy( int idxCopy, jMtx &Uwym, jMtx &Yodp, int idxFE=0 ); // U[Nu,len], Y[Ny,len]
        int virtual StepResponse( int nrwe, double skala, double atU, jMtx &Yodp, int NdlyShift=0, int idxFE=0 ); //Y[Ny,len]

        #define STAN_ON_START            0
        #define STAN_ON_HOT_RESTART1
        #define STAN_STEP_BY_STEP        2
        int virtual OdtworzStan( int typOdtwarzania );


        virtual PtTreeItem_t* btvCfg( cTv *tv, PtTreeItem_t *L1, int childnext );
        virtual PtTreeItem_t* btvData( cTv *tv, PtTreeItem_t *L1, int childnext );
        virtual void printCfg( PtWidget_t *inf=NULL );
```

```
        void virtual printData( PtWidget_t *inf=NULL );

        // pozostale funkcje i dane sa nieistotne w tym raporcie
};
```

Nazwy funkcji nawiązują do typowych usług jak odpowiedz skokowa, symulacja, itd.

Należy zwrócić uwagę, że wszystkie funkcje są deklarowane jako wirtualne.

Studium przyjętych argumentów dla obiektu abstrakcyjnego nie jest istotne dla innych użytkowników niż regulator predykcyjny. Do przekazywania danych stosowane są bowiem obiekty skonstruowane specjalnie dla potrzeb regulatora.

Szerszy opis, przeznaczony dla deweloperów znajduje się w pliku z kodem źródłowym modeller_mo4.c.

Case study:

Poniższe studium przypadków przedstawia wybrane realizacje klas pochodnych.

cMOL2

Klasa cMOL2 jest realizacją wielowymiarowego modelu liniowego. Podstawą do jej konstrukcji jest specjalna klasa cMOL.

```
//---------------------------------------------------------------------------
class cMol2 : public cOB2Generic{
public:
        cMol2( int idxCfg, PtWidget_t *errinfo, PtWidget_t *info=NULL );
        cMol2( char onlyCfg, int idxCfg, PtWidget_t *errinfo, PtWidget_t *info=NULL );

        // <inherited>
        // ...

        // class-specific
        sMolCfg     *cfg;
        cMol        *MO;
        jMtx        *Ytmp;
        cMol        *MOCopy;
};
```

cMoModeller4

Klasa jest zaprojektowana w celu dostarczenia pomostu pomiędzy stroną regulatora predykcyjnego (obiektową) a stroną modułu centralnego (nieobiektową). Poszczególne usługi (w postaci szkieletów funkcji) zostały zebrane w osobnym pliku w języku C (modeller_mo4.c), tak aby specyfika języka C++ nie interferowała z pozostałymi modułami. Konieczność stworzenia takiego projektu wynika z przyjętych założeń Z4,Z5, a także Z2,Z3.

Treść funkcji musi odwoływać się do warstwy wyższej która wykonuje niezbędne obliczenia i powinna zostać wypełniona przez właściwe osoby zajmujące się np. modelem nieliniowym. Strona regulatora predykcyjnego nie ogranicza ani nie narzuca sposobu wywołań (bezpośrednie, via moduł centralny, via klient-server).

```
        //---------------------------------------------------------------------------
        class cMoModeller4 : public cOB2Generic{
        public:
                cMoModeller4( int idxCfg, PtWidget_t *errinfo, PtWidget_t *info=NULL );
                cMoModeller4( char onlyCfg, int idxCfg, PtWidget_t *errinfo, PtWidget_t *info=NULL );

                // <inherited>
                // ...

                // class-specific
                // <puste>
        };

        // modeler_mo4.c  C language file
        int  mo4Create( int _idxCfg );          // uwaga + idxFE/additionalCfg !!!
        void mo4GetOBInfo( sMOInfo *_MOInfo );
```

```
int  mo4CreateOnlyCfg( int _idxCfg );
void mo4Destroy( void );
void mo4GetOBInfo( sMOInfo *_MOInfo );
int  mo4SetU( double **U );            // jednoprobkowy, rzeczywisty-wysyla sterowanie, model-wykonuje jeden krok
int  mo4GetLastU( double **U );        // jednoprobkowy, pobiera ostatnio wdrozone, wykonuje jeden krok
int  mo4GetY( double **Y );            // jednoprobkowy, ostatni pomiar
int  mo4Simulate( int len, double **Uwymuszenia, double **Yodp, int idxFE ); // U[Nu,len], Y[Ny,len]
int  mo4SimulateCopy( int idxCopy, int len, double **Uwym, double **Yodp, int idxFE ); // U[Nu,len], Y[Ny,len]
int  mo4StepResponse( int nrwe, double skala, double atU, int len, double **Yodp, int NdlyShift, int idxFE ); //Y[Ny,len]
int  mo4OdtworzStan( int typOdtwarzania );
void mo4PrintCfg( void );              // na stderr
void mo4PrintData( void );             // na stderr
```

Argumentami w większości przypadków są tablice wskaźników do typu double (double **U). W chwili wywołania usługi przez regulator, wskazują one na zaalokowane już przez regulator obszary pamięci o prawidłowych rozmiarach. Ich rozmiary są zależne od wymiarowości modelu, pożądanej długości odpowiedzi skokowej czy też długości podawanej przy wywołaniu symulacji. Alokowana przez regulator pamięć jest ciągła.

Argument int idxFE (FutureExtension) jest zarezerwowany na potrzeby tworzenia nowych podwariantów. W początkowych etapach implementacji będzie miał wartość zero (ustawienie: IDX_DEFAULT). W miarę potrzeb model może być rozszerzany o kolejne podwarianty oznaczane kolejnymi liczbami całkowitymi.

Funkcje mają zwracać zero (sukces) lub kod błędu.

Dalsze szczegóły znajdują się w pliku źródłowym modeller_mo4.c.

cOB2Boss

Klasa cOB2Boss pełni analogiczną funkcję jak klasa cMoModeller4. Jej głównym zastosowaniem jest dostarczenie pomostu pomiędzy regulatorem, a systemem centralnym.

Ze względu na pełnioną funkcję, istotne są usługi:

```
int  mo4SetU( double **U );            // jednoprobkowy, rzeczywisty-wysyla sterowanie, model-wykonuje jeden krok
int  mo4GetLastU( double **U );        // jednoprobkowy, pobiera ostatnio wdrozone, wykonuje jeden krok
int  mo4GetY( double **Y );            // jednoprobkowy, ostatni pomiar
```

przeznaczone do współpracy z rzeczywistym obiektem.

Usługi:

```
int  mo4Simulate( int len, double **Uwymuszenia, double **Yodp, int idxFE ); // U[Nu,len], Y[Ny,len]
int  mo4SimulateCopy( int idxCopy, int len, double **Uwym, double **Yodp, int idxFE ); // U[Nu,len], Y[Ny,len]
int  mo4StepResponse( int nrwe, double skala, double atU, int len, double **Yodp, int NdlyShift, int idxFE ); //Y[Ny,len]
int  mo4OdtworzStan( int typOdtwarzania );
```

tracą sens i ich treść powinna być pusta.

Uwaga:

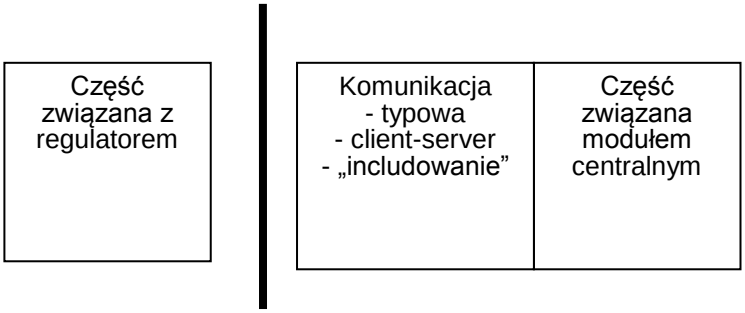
Zgodnie z tematem realizowanego przez autora zadania, raport przedstawia projekt wielowymiarowego modelu z punktu widzenia regulatora predykcyjnego. Inne aspekty związane z modelem (modelowanie, identyfikacja, optymalizacja), stanem obiektu (odtwarzanie stanu, pomiary), a także komunikacja (klient/server, driver) znajdują się poza zakresem prac.

3.4. Implementacja przedprototypowe i stanowisko testowe

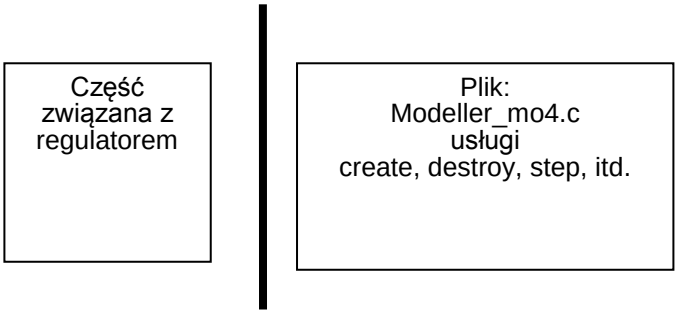
Poniżej przestawiono przedprototypową implementację aplikacji przeznaczonej do testowania modeli. Stanowi ona pewien kod techniczny ułatwiający testowanie i w ostatecznej wersji projektu zostanie usunięta.

Fragment znajdujący się w całym systemie INSTEPRO zdekomponowano na dwie główne części (patrz rys. 3.5). Po lewej stronie znajdują się obiekty związane z wielowymiarowym modelem liniowym i regulatorem. Po prawej – moduł centralny INSTEPRO, obejmujący w kontekście tego raportu komunikację, synchronizację, a także np. model nieliniowy.

Taki podział pozwala na rozdzielenie wykonywanych zadań. Jest realizacją idei enkapsulacji. Pozwala na redukcję strat czasu podczas realizacji, gdyż poszczególne strony nie muszą wnikać w specyfikę kodu strony przeciwnej. *Założenia: poprawność i idealność projektu modułu centralnego. Założono, że działa.*



Rys. 3.5. Podstawowy poziom dekompozycji



Rys. 3.6. Podstawowy poziom dekompozycji – podział organizacyjny

Metodologia implementacji:

- Implementacja wstępna – obejmuje realizację modułów w samodzielnej aplikacji. Uzasadnienie: pozwala na realizację, bez konieczności wikłania się specyfikę innych modułów.

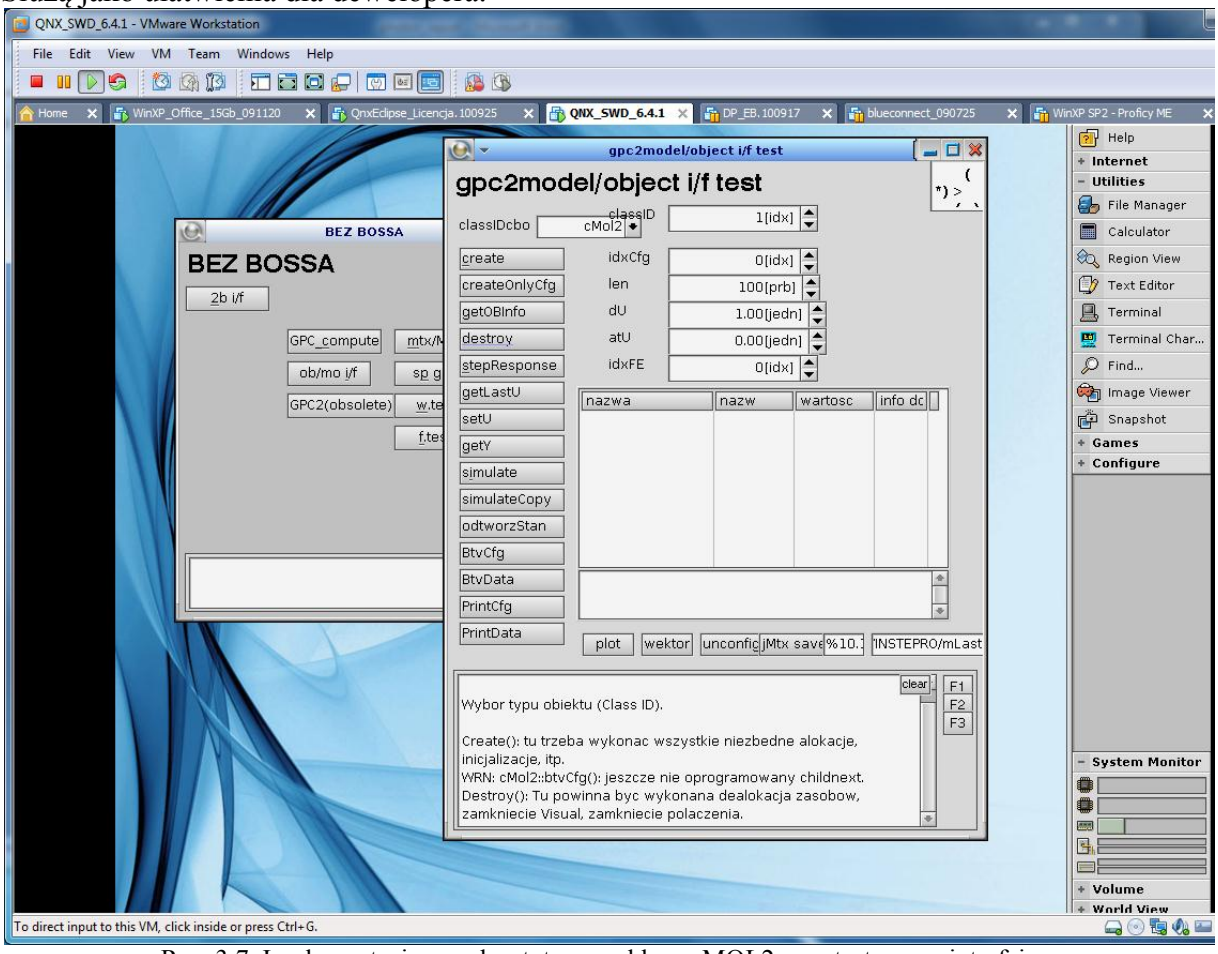
Przyjęto, że będzie realizowana w środowisku Photon. Szacowana zawartość modułu to kilkadziesiąt obiektów, z których każdy zawiera po kilkanaście zmiennych oraz tablic danych o rozmiarach zależnych od wymiarowości obiektu, długości odpowiedzi, itp. Łącznie, objętość danych szacowana jest na kilkastet kilobajtów. Obiekty będą alokowane operatorem new i zarządzane poprzez wskaźniki.
Stosowanie techniki polegającej na pakowaniu danych w jeden ciągły obszar pamięci jest oczywiście możliwe. Jednakże w praktyce, przy skomplikowanej strukturze wewnętrznej modułu, obliczanie pozycji danych prowadzi do konieczności używania w wielu miejscach kodu wyrażeń obliczających adres na podstawie setek danych i ich bieżących rozmiarów. Prowadzi to do dodatkowej, niepotrzebnej komplikacji, zwiększenia ilości kodu źródłowego i ewentualne korzyści zostają przysłonięte przez znacznie większą błędogenność takiej realizacji.

- Weryfikacja implementacji wstępnej
- Integracja z modułem centralnym
- Weryfikacja implementacji
- Opcjonalnie: możliwe jest dezaktywowanie wszystkich okien modułu, co jest równoważne odcięciu użytkownika (człowieka) od interakcji z modułami. Satysfakcjonuje to ideę modułu numerycznego bez interfejsu operatora. Można powiedzieć, że idea rozcinania numeryki od wizualizacji jest przemyślana i wciąż zachowana. Jednakże, wymagana jest

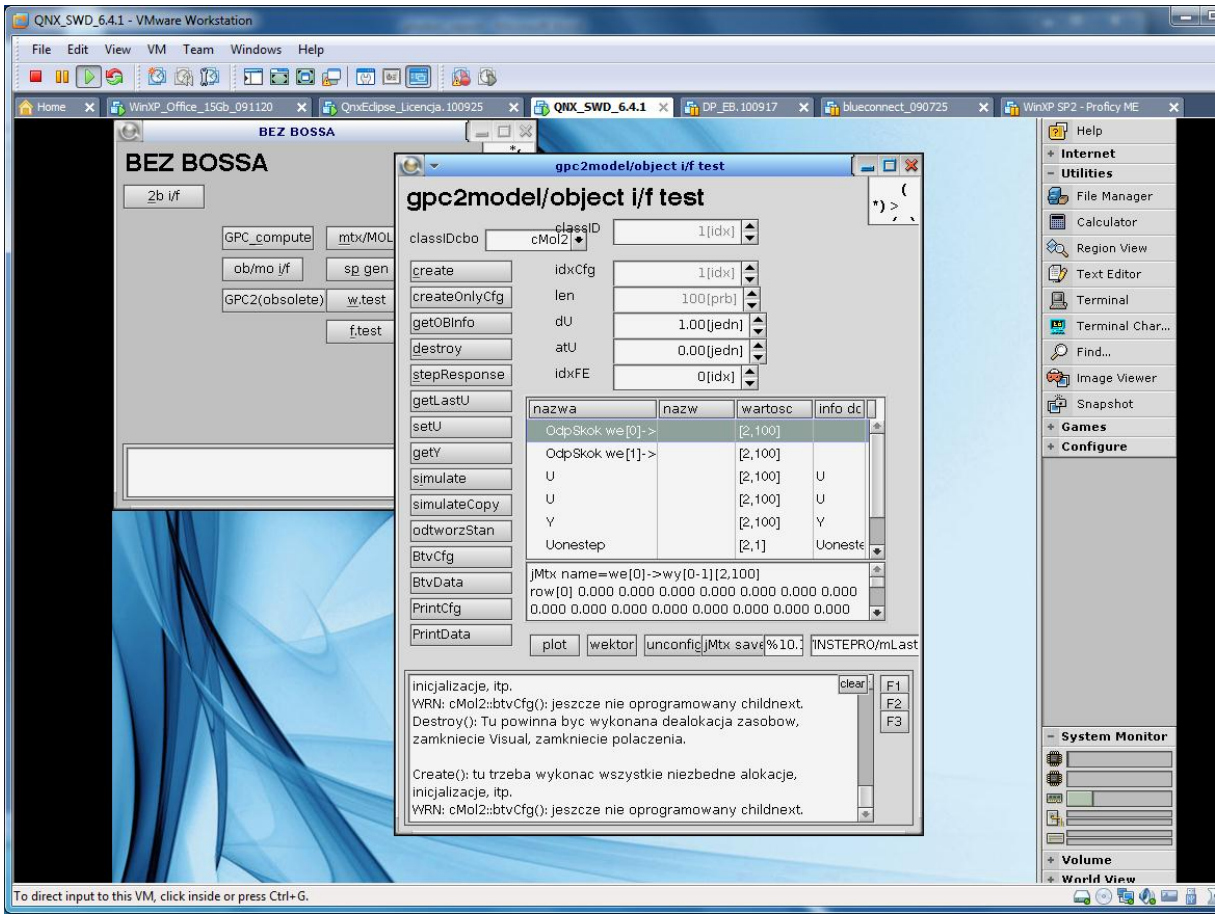
wtedy realizacja osobnego modułu wizualizującego oraz projekt i realizacja pakowania/rozpakowywania różnych danych w różnych kombinacjach. Kwestia wykonania tej dodatkowej pracy pozostaje otwarta.

- Instalacja w systemie rzeczywistym

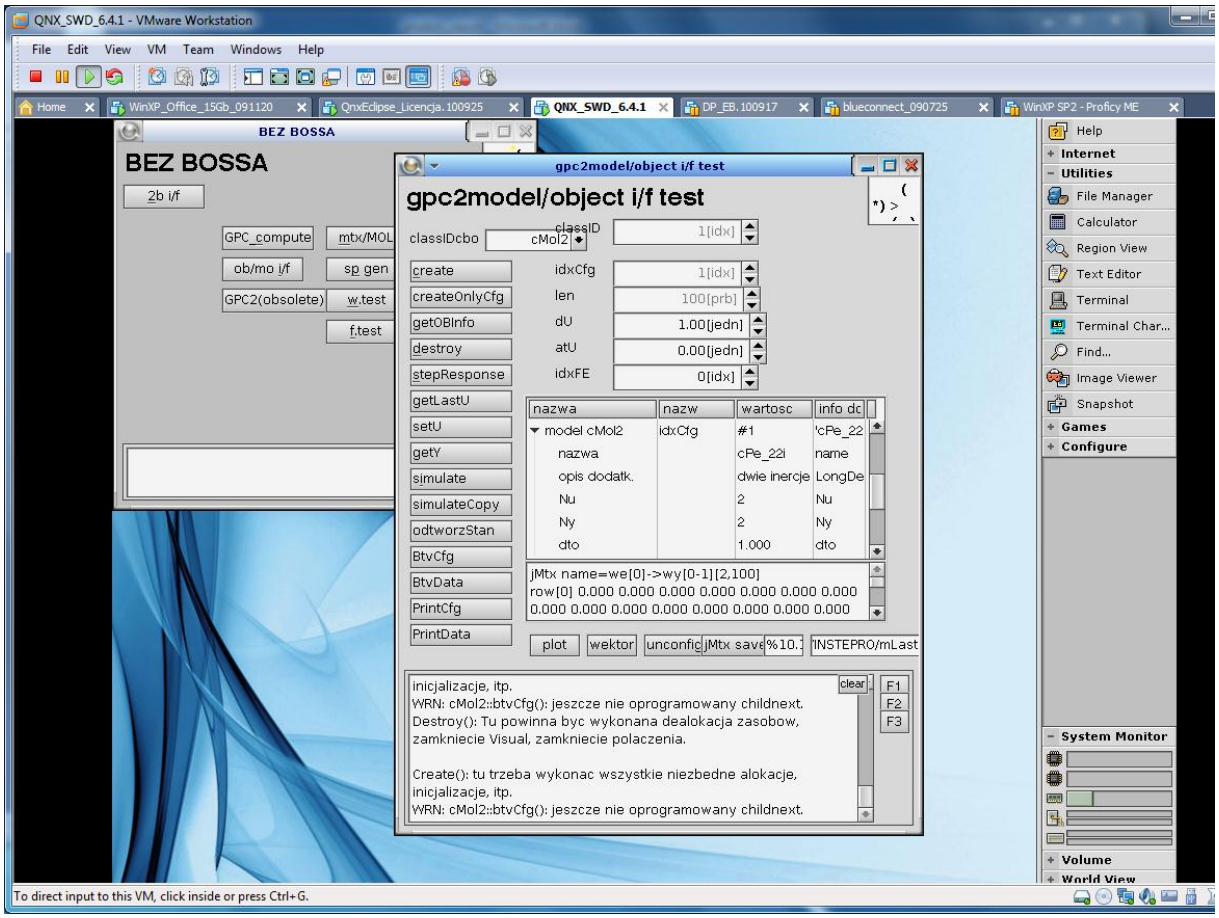
Ilustrowane na rys. 3.1-3.4 w postaci strzałeczki (drucika) powiązania modułów zostały wstępnie zrealizowane. Zastosowano specyfikację opisaną w pkt. „projekt". Przedprototypową implementację przedstawiono na rys 3.7-3.11. Ilustruje ona ideę i należy traktować ją jak pewien pomocniczy kod techniczny weryfikujący koncepcję. W docelowej wersji zostanie usunięta. Na rysunku widoczne są krótkie komentarze wyświetlane po wybraniu danej usługi. Opisują one najważniejsze funkcje danej usługi, jej parametry, itp. Służą jako ułatwienia dla dewelopera.



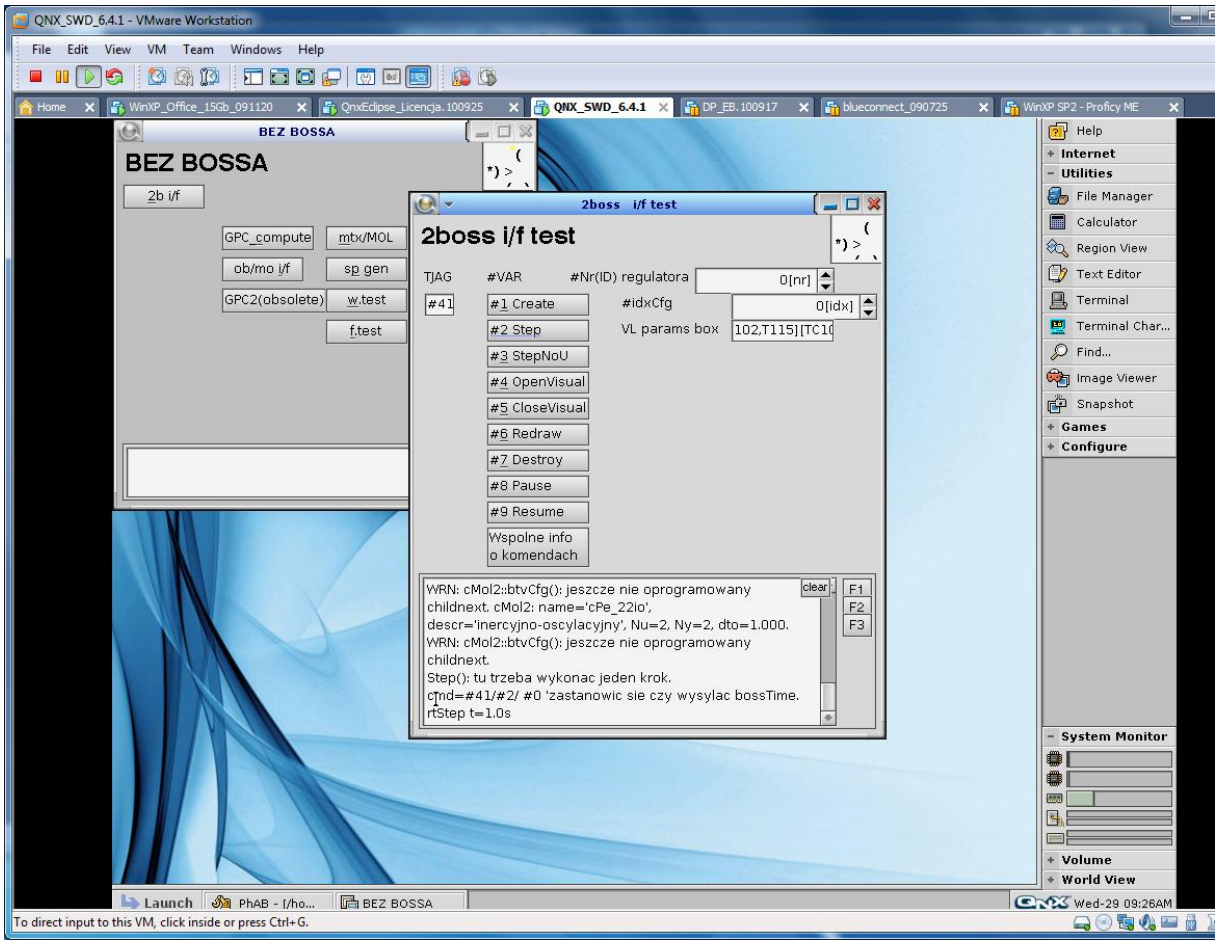
Rys. 3.7. Implementacja przedprototypowa klasy cMOL2 oraz testowego interfejsu



Rys. 3.8. Implementacja przedprototypowa. Widoczne zmienne powiązane z modelem (odpowiedzi skokowe, wektory wymuszenia, odpowiedzi)

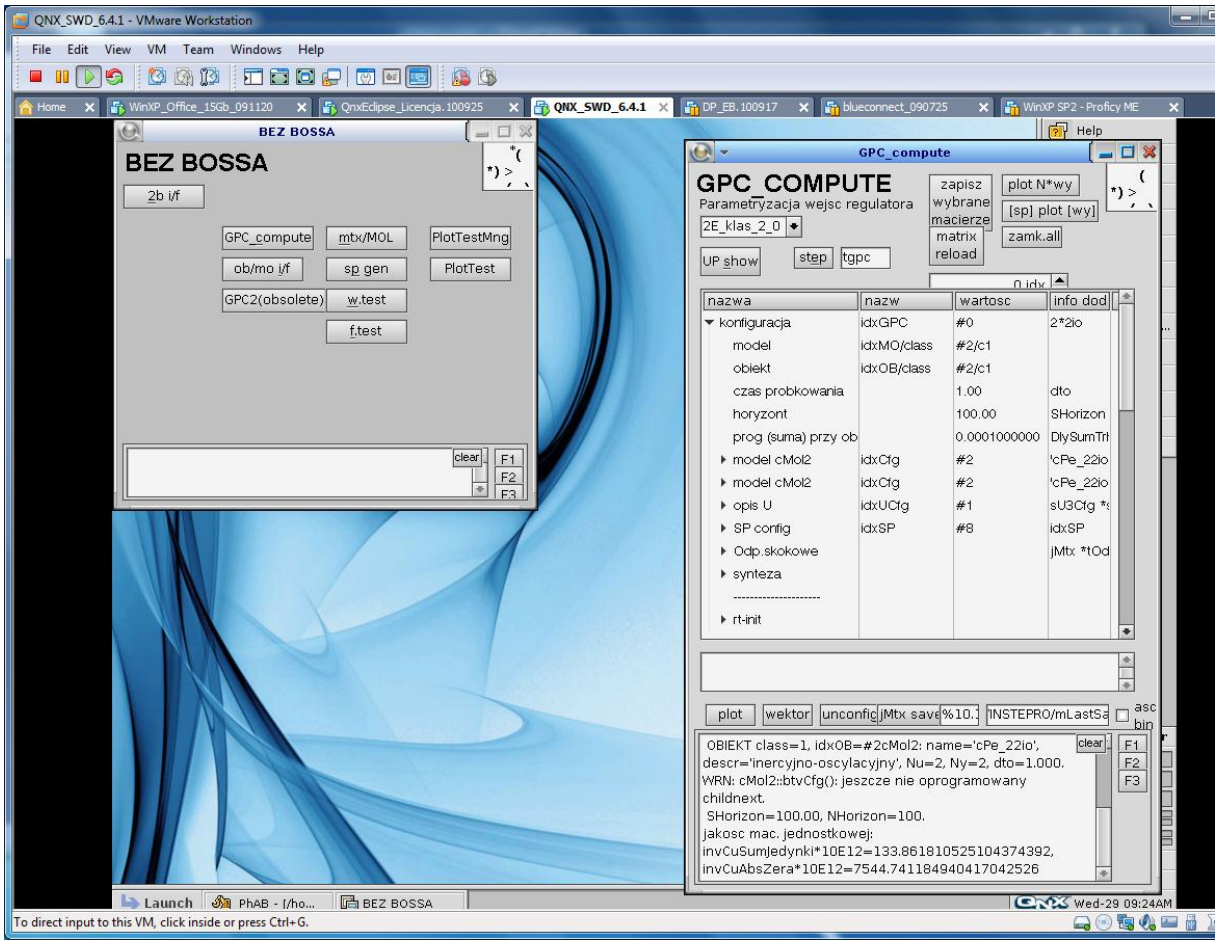


Rys. 3.9. Implementacja przedprototypowa. Widoczna część właściwości modelu (nazwa, ilość wejść, wyjść, czas próbkowania).



Rys. 3.10. Implementacja przedprototypowa testowego interfejsu do regulatora predykcyjnego.

Analogicznie do poprzednich przykładów, zaprojektowano pomocniczy kod techniczny służący do późniejszego testowania regulatora predykcyjnego.



Rys. 3.11. Implementacja przedprototypowa testowego interfejsu wizualizującego regulator predykcyjny.