INNOWACYJNA GOSPODARKA
NARODOWA STRATEGIA SPÓJNOŚCI

Inwestujemy w Waszą przyszłość

INSTEPRO
Zintegrowane Sterowanie Produkcją

UNIA EUROPEJSKA
EUROPEJSKI FUNDUSZ ROZWOJU REGIONALNEGO

Raport wewnętrzny projektu InStePro

Nr 4.3: Projekt symulatora dynamicznych procesów nieliniowych

Data

30.09.2010

Przygotował zespół:

J.T. Duda
T. Pełech-Pilichowski

1. Symulowanie procesów[1]

Wprowadzenie

Zapewnienie wysokiej efektywności i niezawodności nowoczesnych algorytmów regulacji wymaga implementacji algorytmów symulacji procesów opartej na nieliniowych modelach teoretycznych procesu. Modele takie muszą być z natury ściśle dostosowane do specyfiki procesu, co implikuje konieczność włączenia do systemu sterowania na ogół dość złożonego kodu procedur modelujących.

Konstrukcja niezawodnych i łatwych do stosowania technik konstruowania oprogramowania tych procedur pozwala na minimalizację zakresu przetwarzania danych opartego na dedykowanych procedurach modelujących proces (a przez to – zwiększenie niezawodności podsystemu modelowania) oraz zapewnienie wymaganej elastyczności strukturalnej i modyfikowalności procedur modelujących.

Projektowanie oraz implementacja takiego oprogramowania ma na celu zapewnienie maksymalnego wykorzystania uniwersalnych narzędzie wspomagających proces projektowania (CASE), jednak – w odniesieniu do systemów czasu rzeczywistego – ich stosowanie do implementacji modeli matematycznych procesów ciągłych wydaje się jednak dość trudne z następujących przyczyn:

- modele dynamiki procesu zawierają obok stosunkowo prostych (na ogół) funkcji stanu, złożone algorytmy rozwiązywania nieliniowych równań algebraicznych wyjść i stanu ustalonego, które trudno zbudować przy pomocy uniwersalnych narzędzi CASE;
- powiązania modeli matematycznych z otoczeniem procesowym (systemem sterowania komputerowego) są często bardzo złożone, szczególnie jeśli zakłada się możliwość restrukturyzacji (w czasie rzeczywistym) systemu sterowania;
- modele procesów są często prototypowane w środowisku MATLAB-SIMULINK, zatem wydaje się celowe dążenie do stworzenia możliwości łatwego przenoszenia praktycznie gotowego kodu procedur stworzonych dla tej platformy, w środowisko software'owe systemu sterowania.

Poniżej przedstawiono propozycję modułu oprogramowania pisanego w języku C. Zakłada się, że poszczególne pliki stanowią autonomiczne moduły, sprzężone z systemem sterowania poprzez dane konfigurujące, które określają powiązania zmiennych i równań modelu ze zmiennymi procesowymi. Dane konfigurujące wpisuje się w trybie dialogowym przy pomocy odpowiedniego oprogramowania, kontrolującego poprawność syntaktyczną, logiczną i merytoryczną podsystemu. Podsystem modelowania funkcjonuje jako element scentralizowanego systemu obliczeniowo-decyzyjnego realizującego funkcje sterowania nadrzędnego

Modele procesów w wielopoziomowych strukturach sterowania

Systemy komputerowego sterowania złożonymi procesami mają strukturę wielopoziomową i wielowarstwową. Wyodrębniony w takiej strukturze obiekt sterowania jest obsługiwany przez algorytmy optymalizacji statycznej, sterowania dynamicznego i nadzorowania (zwane warstwami), dla których wymagania (w formie ograniczeń i nadrzędnego wskaźnika jakości) formułowane są na wyższym poziomie sterowania (zwanym poziomem zarządzania), a poziom niższy stanowi system wykonawczy.

Z formalnego punktu widzenia wyodrębnienie systemu wykonawczego polega na podzieleniu wektora stanu procesu na dwa podwektory x_u (dynamika względnie wolnego obiektu sterowania z abstrakcyjnymi wejściami u) i x_m (dynamika szybszych podprocesów realizujących decyzje sterujące u poprzez układy sterowania wykonawczego generujące wejścia manipulowane m). Wysoką jakość i niezawodność sterowania można osiągnąć tworząc elastyczne struktury przełączalnych i adaptowalnych regulatorów,

[1] Materiał sporządzono na podstawie opracowania J.T. Duda „*Modelowanie i symulacja procesów w systemach komputerowego sterowania nadrzędnego*", Katedra Automatyki AGH w Krakowie

których struktura (podział na obiekty sterowania i układy wykonawcze) może być zmieniana w zależności od wiarygodności pomiarów i stanu procesu. Dla każdej z takich alternatywnych struktur układu sterowania muszą być dostępne modele procesów objętych pętlami regulacji.

Etapem poprzedzającym implementację powyższej koncepcji sterowania jest definicja drzewa procesów, którego pień stanowi proces główny opisany rozbudowanym modelem nieliniowym w przestrzeni stanu, zawartym w dedykowanym pliku źródłowym przypisanym do całej rodziny procesów. Plik ten musi zawierać również dedykowane procedury obliczania stanu ustalonego oraz formuły nadrzędnego wskaźnika jakości.

Podzbiorami procesu głównego są podprocesy o ograniczonym rzędzie i odpowiednio zdefiniowanych wejściach i wyjściach. Możliwości oddziaływania na proces specyfikuje się definiując zbiór pętli regulacyjnych SISO z uwzględnieniem wszystkich alternatywnych możliwości regulacji. Poszczególne pętle zawierają proste algorytmy regulacji, które mogą być uaktywnione w przypadku braku innych możliwości lub zbyt słabej efektywności sterowania innymi metodami.

Elastyczne struktury regulacji wielowymiarowej można zdefiniować poprzez specyfikację podprocesów. Każdy podproces zawierający sterowane wejścia i wyjścia przewidziane do stabilizacji może być obiektem sterowania z przyporządkowanym zbiorem algorytmów regulacji. Definiując odpowiednio podprocesy można stworzyć wielopoziomową strukturę kaskadową złożoną z regulatorów MIMO i SISO na poziomach wyższych oraz SISO na poziomie sterowania bezpośredniego.

Dla zdefiniowanego jak wyżej obiektu sterowania winny być wykonywane odpowiednie usługi, takie jak: rozpoznawanie stanu procesu, linearyzacja modelu do postaci stacjonarnych i niestacjonarnych macierzy stanu, macierzy transmitancji lub odpowiedzi skokowych, rekurencyjna korekcja modeli transmitancyjnych, obliczanie macierzy wagowych wskaźnika jakości, obserwacja stanu, symulacja. Optymalizacja statyczna winna być realizowana tylko dla całych procesów głównych dla uwzględnienia interakcji między podprocesami. Jej wyniki wykorzystuje się do linearyzacji modeli wszystkich podprocesów, obliczania macierzy wagowych wskaźnika jakości itp.

Ilościowe i jakościowe oceny przebiegu procesu są dokonywane na podstawie odpowiednio wyspecyfikowanych zmiennych procesowych. Zmienne te dzieli się na wielkości wejściowe sterujące, wejściowe zakłócające i wyjściowe. Wejścia zakłócające i wyjścia mogą być bezpośrednio mierzalne lub obliczane na podstawie innych wielkości. Wszystkie zmienne procesowe są rejestrowane w bazach danych systemu (dostępne są zarówno ich wartości bieżące jak i przeszłe). Obok zmiennych procesowych w modelach procesów mogą być wykorzystywane zmienne lokalne (nie wykorzystywane dla potrzeb nadzorowania, wizualizacji itp.).

Konstruując środowisko software'owe do elastycznego sterowania w strukturach wielopoziomowych należy przyjąć następujące założenia:

1. Równania stanu procesu mają postać nieliniową zależną od specyfiki procesu, przy czym są one zapisane jawnie dla kolejnych zmiennych stanu.
2. Celowe jest formułowanie alternatywnych równań stanu i równań wyjść reprezentujących różne warianty modelu.
3. Równania wyjść mogą mieć postać uwikłaną.
4. Kwalifikacja zmiennych modelu do zbioru wejść, wyjść czy wektora stanu zależy od miejsca rozważanego procesu w strukturze wielopoziomowej regulacji oraz przyjmowanych uproszczeń.
5. Wszystkie zmienne wejściowe i wyjściowe modelu są zmiennymi prostymi i muszą być przypisane do zmiennych procesowych systemu, przy czym zmiennej modelu może odpowiadać zbiór zmiennych procesowych, reprezentujących redundantne zmienne pomiarowe.

6. Zmienne stanu mogą być zmiennymi lokalnymi modelu lub zmiennymi wyjściowymi. W pierwszym przypadku mogą być pogrupowane w tablice (zmienne indeksowane).
7. Wszystkie zmienne wejściowe muszą być explicite wykorzystywane w równaniach modelu, podobnie jak zmienne stanu niebędące wyjściami.
8. Zmienne stanu nie mogą być podstawiane w funkcjach stanu.
9. Model winien uwzględniać zależność opóźnień transportowych od zmiennych procesowych.

Integracja nieliniowych modeli matematycznych z systemem sterowania

Oprogramowanie modeli matematycznych jest często budowane w wersji prototypowej i testowane w języku MATLAB, który dostarcza bogatych możliwości graficznej prezentacji właściwości obiektu oraz analizy różnych struktur sterowania. W omawianej dalej koncepcji zakłada się, że uzyskaną tą drogą kod winien być zapisany w języku C z zachowaniem nazewnictwa zmiennych i minimalną modyfikacją instrukcji kodujących równania modelu. Przyjęto następujące zasady:

- Wszystkie procedury modelujące rodzinę procesów są umieszczone w jednym pliku źródłowym, a co najmniej tworzą jeden plik binarny
- Zmienne modelu dzieli się na cztery grupy, tj.:
  a) zmienne procesowe jako zmienne proste globalne dla całego systemu;
  b) lokalne zmienne stanu będące zmiennymi prostymi lub tablicami;
  c) parametry modelu;
  d) zmienne robocze służące do przesyłania pewnych danych między funkcjami modelu.
- Zmienne (a) i (b) są typu double i zajmują spójny obszar pamięci alokowany dynamicznie i adresowany nazwą `arg[]` umieszczaną na liście parametrów formalnych procedur przetwarzających. Żadna z tych zmiennych nie jest przekazywana do żadnej funkcji jawnie poprzez wartość, a operacje wykonywane są na nich przez odwołanie do zawartości adresu odpowiadającego danej zmiennej w tablicy `arg[]`.
- W celu zachowania czytelności kodu, wewnątrz procedur powyższe odwołania realizuje się poprzez nazwy lokalne przyjęte w wersji prototypowej i definiowane instrukcjami preprocesora z jawnym przypisaniem indeksu w tablicy `arg[]`. Przykładowo, jeśli w modelu zastosowano nazwy `Twe`, `Fwy`, `Fwe`, `Hkol`, `Xkol[20]`, `X_R`, gdzie `Xkol` jest tablicą, a pozostałe zmiennymi prostymi, przy czym `X_R` jest tożsama z `Xkol[19]` to wprowadza się następujące definicje:

```
#define Twe  (*(arg))     // temperatura strumienia wlotowego
#define Fwy (*(arg+1))    // przepłw wyjściowy cieczy ze zbiornika
#define Hkol(*(arg+2))    // poziom w zbiorniku
#define Xkol  (arg+3)     // sklad cieczy na polkach kolumny
#define Fwe (*(arg+23))   // przepływ wyjściowy ze zbiornika
#define X_R (*(arg+22))   // sklad cieczy na ostatniej polce
```

  Indeks nazwy występującej bezpośrednio po nazwie tablicy determinuje rozmiar tej tablicy (nazwa `Fwe` po `Xkol`). Jak widać, dopuszcza się przypisanie różnych nazw do tego samego miejsca w pamięci, co pozwala m.in. nadać wybranym elementom wektora stanu status zmiennej procesowej wyjściowej (np. `X_R≡Xkol[19]`).

  Powyższe definicje uniemożliwiają omyłkowe zamaskowanie globalnych nazw modelu deklaracjami lokalnymi. Jeśli zachodzi potrzeba, powyższe definicje mogą być anulowane na końcu pliku przy pomocy instrukcji `#undef`.

- Parametry modelu oraz zmienne robocze tworzą odrębne struktury danych specyficzne dla modelowanego procesu, których prototypy są zamieszczone w pliku nagłówkowym.

- Każdej funkcji stanu odpowiada oddzielna funkcja języka C, której wyjście jest przyrostem zmiennej stanu przypisanej do tej funkcji. Funkcje wyjść mogą obliczać wielkości wyjściowe w formie wyjścia funkcji lub przez podstawienia wewnątrz funkcji.

  Wszystkie funkcje stanu i wyjść mają jednakową listę parametrów formalnych.

```
double (*state_fun)(shint fun_no, double arg[], char *par, char *buf);
double (*outp_fun)(shint outp_no, double arg[], char *par, char *buf);
```

  gdzie `fun_no`, `outp_no` oznaczają indeks funkcji w systemie; `arg[]` - tablicę zmiennych modelu; `*par` - adres struktury parametrów; `*buf` - adres struktury zmiennych roboczych.

- Zmienne stanu nie mogą być jawnie podstawiane ani w funkcjach stanu ani wyjść. Mogą być natomiast modyfikowane (dla uwzględnienia silnych nieliniowości, np. ograniczeń) przez jedną funkcję *H()* wywoływaną po modyfikacji zmiennych stanu przyrostami ciągłymi. Funkcja ta jest deklarowana ogólnie jako:

  ```
  void (*set_parameters)(double arg[], char *par, char *buf);
  ```

  Kod przykładowej funkcji stanu ma postać:

```
#define PAR (*d)
#define OBL (*w)
// ------------------------------------------------------------------
double poziom(shint nr_stanu, double arg[], char *par, char *buf)
 {struct par_wyp *d; struct wyniki_wyp *w;
  double dH;
  // ------------------ Cialo procedury ---------------------------
  d=(struct par_wyp *)par; w=(struct wyniki_wyp *)buf; //podstawienia
  dH=(Fwe*PAR.ro_we-Fwy*OBL.rob)/(PAR.So*OBL.rob);  //rownanie stanu
  return dH;
 }
// ------------------------------------------------------------------
```

  gdzie `So` i `ro` są parametrami, a `rob` – zmienną roboczą.

- Dla każdej rodziny procesów zapisuje się funkcję obliczającą stan ustalony procesu głównego. Wewnętrzna struktura tej funkcji jest dowolna, a jej deklaracja ma postać:

```
shint (*steady_state)(double steady_arg[], shint steady_option, char *par,
 char *buf);
```

Wszystkie dane opisujące model zawarte są w strukturze *Proces* zawierającej m.in.:

a) unikalny numer procesu;
b) numer procesu głównego rodziny;
c) liczbę zmiennych procesowych wykorzystywanych w modelu;
d) liczbę zmiennych lokalnych modelu (rozmiar tablicy *D*)
e) tablicę indeksów (w bazie globalnej) zmiennych procesowych modelu odpowiadających kolejnym elementom tablicy *D*;
f) liczbę wszystkich równań wyjść i liczbę wszystkich równań stanu;
g) kod statusu kolejnych elementów tablicy *D* (sterowanie, zakłócenie, wyjście, stan);
h) liczbę równań stanu włączonych do modelu (rząd modelu);
i) numery równań stanu i indeksy (w tablicy *D*) odpowiadających im zmiennych stanu;
j) liczbę włączonych do modelu funkcji wyjść i indeksy zmiennych wyjściowych obliczanych przez nie jawnie (w formie wyjścia funkcja);
k) liczbę wejść modelu i parametry charakteryzujące każde wejście: indeks zmiennej procesowej, opóźnienie stałe, parametry do obliczania opóźnienia transportowego;
l) rozmiary i adresy bufora roboczego oraz struktury parametrów modelu, adres funkcji *H()* modyfikującej stan i funkcji obliczającej stan ustalony, tablicę adresów funkcji stanu i tablicą adresów funkcji wyjść;

m) adresy obszarów przeznaczonych na modele procesu obliczane w czasie rzeczywistym, tj. odpowiedzi skokowe modelu nieliniowego i liniowego, macierze zlinearyzowanych równań stanu oraz parametry modeli transmitancyjnych;
n) nazwę omawianego tu pliku źródłowego kodującego model dynamiki.

Do wpisywania powyższych danych opracowano dialogowy program konfigurujący, pozwalający na wybór zmiennych procesowych modelu spośród zmiennych procesowych systemu sterowania, przypisanie im zmiennych mnemonicznych wykorzystywanych w procedurach modelujących z podziałem na wejścia i wyjścia, wybór funkcji stanu i wyjść, przypisanie im zmiennych modelu oraz wpisanie wymaganych parametrów wejść.

Dla umożliwienia łatwego wprowadzania tych danych poprzez wybór z menu, opracowano program analizujący kod źródłowy procedur modelujących. Przygotowuje on listę nazw zmiennych modelu oraz funkcji stanu i wyjść, sprawdza krotność użycia tych zmiennych w funkcjach stanu i wejść sygnalizując alarmem wystąpienie podstawień zmiennych wybranych jako zmienne stanu.

System sygnalizuje obecność zmiennych, którym nie przypisano zmiennej procesowej (alarmując takie sytuacje w odniesieniu do wejść), a także brak podstawień zmiennych wyjściowych. Uniemożliwia nieprawidłowe klasyfikacje zmiennych (np. przypisanie statusu wejścia zmiennej tożsamej z pewną zmienną stanu), wymusza poprawne logicznie definiowanie opóźnień transportowych.

Program konfigurujący modele procesów korzysta z danych konfigurujących system sterowania nadrzędnego (nazwy i indeksy zmiennych, specyfikacje pętli regulacyjnych). Dla każdego procesu (zestawu zmiennych, równań stanu i wyjść) dane konfigurujące model tworzą jeden rekord struktury *Proces*.

Struktura modelu może być łatwo modyfikowana przez usunięcie zmiennej lub zmianę jej statusu, przy czym blokowane są modyfikacje niedopuszczalne ze względu na spójność logiczną modelu.

Po zdefiniowaniu procesu głównego można łatwo utworzyć proces potomny (np. o mniejszym rzędzie, mniejszej liczbie wejść lub wyjść). Proces potomny może operować tylko na podzbiorach zmiennych procesowych przypisanych do procesu macierzystego i dziedziczy globalne atrybuty tych zmiennych (indeksy, nazwy, współczynniki skali). Inny może być natomiast status poszczególnych zmiennych (wejście, wyjście, stan) i inny zestaw funkcji stanu oraz wyjść.

Wykorzystując atrybuty zmiennych modelu zdefiniowane dla całego sytemu sterowania specyfikuje się algorytmicznie wszystkie pętle regulacyjne, które mogą być utworzone dla danego procesu o przyjętej strukturze. Plik kodujący model procesu musi zawierać funkcję typu:

```
shint (*model_start)(void *P, shint *outp_dim);
```

w której należy przypisać explicite odpowiednim elementom struktury *Proces* nazwy (adresy): procedury obliczającej stan ustalony, procedury *H()* modyfikującej stan oraz wpisać nazwy funkcji stanu i funkcji wyjść do zadanych elementów tablicy adresów funkcji modelu. Dla tablicy adresów funkcji stanu przyjęto obligatoryjną nazwę `state_fun[]`, a dla adresów funkcji wyjść – `outp_fun[]`. Przykładowy kod źródłowy tej funkcji ma postać:

```
shint przypisz_model_wyparki(void *P, shint *outp_dim)
 {shint nst, state_dim;
/* ------------------- Czesc universalna ---------------------------- */
  struct Proces *p;
  double (**state_fun)(shint nr_stanu, double arg[], char *par, char *buf);
  double (**outp_fun)(shint outp_no, double arg[], char *par, char *buf);
  p=(struct Proces *)P;
  state_fun=p->state_fun; outp_fun=p->outp_fun;
/* ------------------- Czesc specyficzna ---------------------------- */
  p->set_parameters=ustaw_param_wyp;
  p->steady_state=stan_ustalony_wyp;
```

```
/* ------------------ funkcje stanu ---------------------------------- */
  state_fun[0]=temp_oleju;
  state_fun[1]=temp_cieczy;
  state_fun[2]=poziom;
  for(nst=3;nst<22;nst++) state_fun[nst]=sklad_kol;
/* ------------------ funkcje wyjsc ---------------------------------- */
  outp_fun[0]=cisn_pary;
  outp_fun[1]=sklad_pary;
  outp_fun[2]=F_pary;
  outp_fun[3]=F_wyjs;
  *outp_dim=4;
  return state_dim=22;
 }
```

Można również przygotować procedurę ustawiającą domyślne wartości parametrów modelu oraz procedurę umożliwiającą modyfikacje parametrów w trybie dialogowym.

Wszystkie pliki modelujące są włączane do kodu źródłowego systemu przy pomocy pliku konfiguracyjnego, w którym podstawia się wartości pól następującej struktury:

```
struct model_par
 {char *Param_File;            // nazwa pliku zawierającego parametry
  shint (*par_rep)(char *r);   // adres funkcji raportującej parametry
  void (*par_default)(char *r);// adres funkcji ustawiającej param.domyślne
  shint (*model_start)(void *P, shint *outp_dim);
  shint par_size;              // rozmiar struktury parametrów
  shint buf_size;              // rozmiar bufora roboczego
};
```

Plik konfigurujący łączy modele wszystkich rodzin procesów i ma postać:

```
//=================== PLIK KONFIGURUJACY SYSTEM ========================
static char *konf_file="/home/jdu/modele/proc_set.c";
#include "/home/jdu/blank/proc_str.h"      // deklaracje struktur systemu
void assign_processes(void);
void zerr_processes(void);
// ========== deklaracje dla wyparki ====================================
#define WYP_MOD "/home/jdu/modele/wyp_mod.c"   // kod zrodlowy modelu
#include WYP_MOD                               // tu wlaczamy kod modelu
//----------- Deklaracja zamkniecia podstawien -----------------------
#define MPROC_DIM 1               // tu wstawia się liczbe rodzin procesow
// ====================================================================
static char *Model_File[MPROC_DIM];
static struct model_par MP[MPROC_DIM];  // definicja struktury w proc_str.h
// ====================================================================
void assign_processes(void)
 {shint proc;
  //=========== Podstawienia dla wyparki ================================
  proc=0;
  Model_File[proc]=WYP_MOD;                     // plik zrodlowy modelu
  MP[proc].model_start=przypisz_model_wyparki;  // procedura startujaca
  MP[proc].Param_File="../modele/par_wyp.dat";  // zbior parametrow
  MP[proc].par_rep=raport_param_wyparki;        // procedura raportu param.
  MP[proc].par_default=ustaw_param_wyparki;     // proc.podstwien param.def
  MP[proc].par_size=sizeof(struct par_wyp);     // rozmiar strukt.param.
  MP[proc].buf_size=sizeof(struct wyniki_wyp);  // rozmiar bufora robocz.
  //----------- Podstawienia dla innych procesow  -------------------------
/*proc=1;
  Model_File[proc]=xxx_MOD;                     // zbior modelu
  MP[proc].Model_start=..nazwa_funkcji....;     // proc.ustawiajaca modele
  MP[proc].Param_File=xxx_PAR;                  // zbior parametrow
  MP[proc].par_rep=.....nazwa_funkcji..;        // procedura raportu param.
  MP[proc].par_size=sizeof(struct .......);     // rozmiar strukt.param.
  MP[proc].buf_size=sizeof(struct ..........);  // rozmiar bufora robocz.
  proc=2;
  ..................... i nastepny */
 }
```

Plik `proc_set.c` jest dołączany do każdego zadania wykorzystującego modele. Zadanie takie, bezpośrednio po uruchomieniu, wywołuje funkcję `assign_processes()` i wczytuje z dysku dane

konfigurujące modele rodzin procesów. Każdemu podprocesowi odpowiada w systemie jeden rekord struktury *Proces*. Umożliwia to dalej wykonywanie operacji na dedykowanych modelach przy pomocy uniwersalnych procedur linearyzacji, symulacji itp. Zapisane w strukturze *Proces* przyporządkowanie zmiennych modelu do zmiennych procesowych systemu pozwala przy tym na automatyczne pobieranie danych wejściowych modelu z baz danych systemu oraz wpisywanie wyników symulacji do tych baz (np. dla potrzeb sterowania predykcyjnego, diagnostyki, alarmowania itp.).

Podsumowanie

Proponowana technika implementacji nieliniowych modeli matematycznych procesów umożliwia łatwe połączenie dedykowanego kodu źródłowego procedur modelujących z systemem sterowania spełniającym typowe funkcje zbierania danych oraz ich standardowego przetwarzania. Odpowiednio oprogramowane zadania symulacji i linearyzacji są łączone z procedurami modelującymi przy pomocy przejrzystego pliku konfigurującego. Modyfikacja modelu procesu wymaga:

a) zmiany kodu źródłowego odpowiednich procedur modelujących i skompilowania;
b) wywołania (w trybie off-line) programu konfigurującego i uzupełnienia lub modyfikacji danych konfigurujących proces (program sprawdza przy tym poprawność formalną i merytoryczną specyfikacji).

W punkcie (b) można zmienić specyfikację modelu lub zdefiniować nowy podproces bez zmiany kodu źródłowego jak w (a) o ile dostępne są wymagane funkcje stanu/wyjść.

Nowe specyfikacje procesów (rekordy struktury *Proces*) muszą być ponownie wczytane we wszystkich zadaniach wykorzystujących modele.

W celu dołączenia do systemu nowej rodziny procesów (rozbudowa systemu sterowania) należy wykonać następujące operacje:

a) stworzenie oddzielnego pliku źródłowego zawierającego zestaw funkcji modelu
b) wpisanie w pliku `proc_set.c` danych kolejnego rekordu struktury `model_par` specyfikujących rodzinę i zwiększenie wartości stałej symbolicznej `MPROC_DIM`
c) rekompilacja programu konfigurującego modele;
d) wywołanie programu konfigurującego w celu wyspecyfikowania procesu głównego rodziny, a następnie procesów potomnych (o ile są wymagane);
e) rekompilacja i ponowne załadowanie wszystkich zadań wykorzystujących modele.