Tendencje rozwojowe środowisk inżynierskich dla systemów sterowania

Leszek Trybus

Plan

- Stan aktualny
- Tendencje programistyczne
- Tendencje *runtime*
- Zastosowania i uzupełnienia

Współpracownicy:

D. Rzońca, J. Sadolewski, A. Stec, Z. Świder, B. Trybus

Politechnika Rzeszowska
Katedra Informatyki i Automatyki

Charakterystyka ogólna

- Standardy
  - IEC 61131-3 – cykl wykonywania programu
    - powszechne zastosowanie
  - IEC 61499 – aktywacja zdarzeniowa
    - nieliczne zastosowania
- Składniki środowiska IEC 61131-3
  - Programowanie
    - IDE: edytory języków, kompilatory, translatory, tryb *online*
    - symulator – niezależny od języka
    - konfigurator komunikacji i I/O
  - *Runtime* – środowisko wykonawcze:
    - platforma docelowa
    - PC (tryb *online*, symulator, *softcontroller*)

Środowiska i rozwiązania

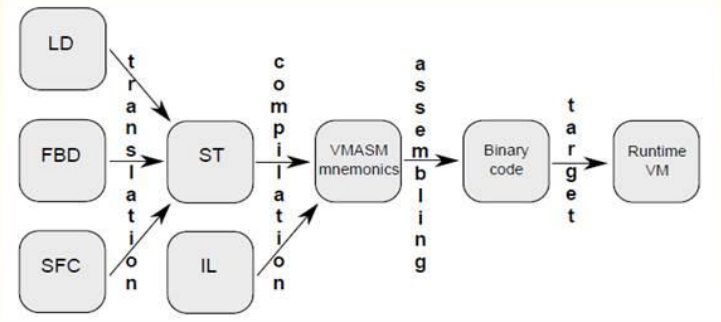
- Środowiska
  - Czołowi producenci – PLC, PAC, DCS
    Step 7, Concept, TwinCAT, Control Builder , …
  - Niezależni – PLC, PAC (dla producentów SMES)
    CODESYS, GEB, STRATON, LogicLab, [ISaGRAF]
  - Inni – AVR, ARM, PC, FPGA
    Beremiz, Academic-4, .NET CLR, CPDev (*framework*)
- Rozwiązania techniczne

kod źródłowy → kod maszynowy	szybkie
Czołowi prod., CODESYS, LogicLab	dedykowane dla CPU
kod źródłowy → C/C++ → kod maszynowy	wieloplatformowe
Beremiz, GEB	narzędzia C/C++
kod źródłowy → język pośredni → maszyna wirtualna *runtime*	wieloplatformowe
[ISaGRAF], STRATON , Acad.-2, .NET CLR, CPDev	narzut VM

Rozwiązanie z maszyną VM

Maszyna wirtualna VM (*virtual machine*) jest procesorem zrealizowanym programowo wykonującym kod pośredni wygenerowany przez dedykowany kompilator na podstawie kodu źródłowego.

- Środowisko bazujące na VM



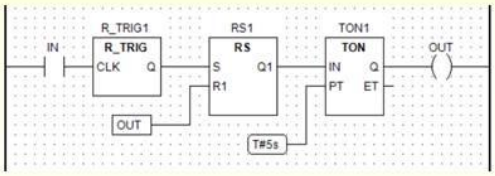
- Wieloplatformowe
  8, 16, 32-bit micro
  PC(x86), FPGA
- Java Virtual Machine
  .NET Common Language Runtime
  [P-code Machine, VDM]

- VMASM – Virtual Machine Assembler (CPDev)
  Inne: TIC, IL, SIL, ? VM napisana w C

Translatory LD, FBD

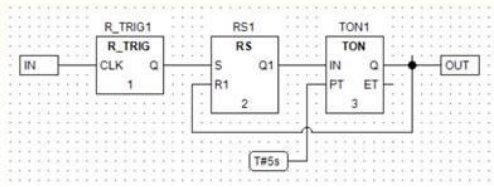
Automatycznie tworzone zmienne lokalne reprezentujące wewnętrzne połączenia. Potrzebne zwłaszcza w trybie *online*.

- LD → ST




```
out_contact_IN_80_90 := IN;
R_TRIG1(CLK:=out_contact_IN_80_90,Q=>out_R_TRIG1_Q);
RS1(S:=out_R_TRIG1_Q,R1:=OUT,Q1=>out_RS1_Q1);
TON1(IN:=out_RS1_Q1,PT:=T#5s,Q=>out_TON1_Q);
OUT := out_TON1_Q;
```

- FBD → ST




```
R_TRIG1(CLK:=IN,Q=>out_R_TRIG1_Q);
RS1(S:=out_R_TRIG1_Q,R1:=out_TON1_Q,Q1=>out_RS1_Q1);
TON1(IN:=out_RS1_Q1,PT:=T#5s,Q=>out_TON1_Q);
OUT := out_TON1_Q;
```

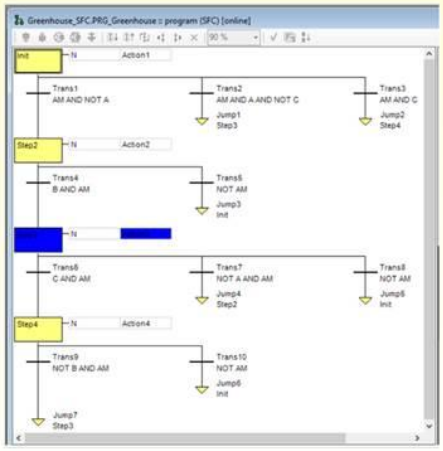
- Edytory LD, FBD – połączenia tworzone automatycznie (algorytm A*)

Rytro 2019 Stan aktualny 21-22 maja

Język SFC

- Edytor – kroki, akcje, tranzycie, kwalifikatory (N, R, S, L, D, …)

Przykład (*online*)

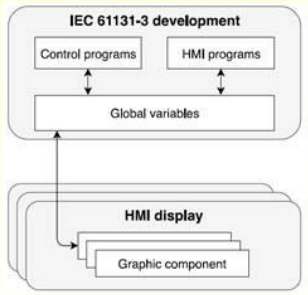


- Krok – Akcja

Kod *next step flags* reprezentuje strukturę drzewa SFC.

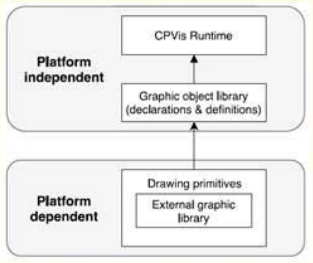
Interfejs HMI

- Współpraca sterowania z HMI



- Wieloplatformowe HMI

TFT, LCD, monitor, …



RamTex, GLCD, GDI+

- Panele operatorskie TFT (Praxis)

Programowanie obiektowe

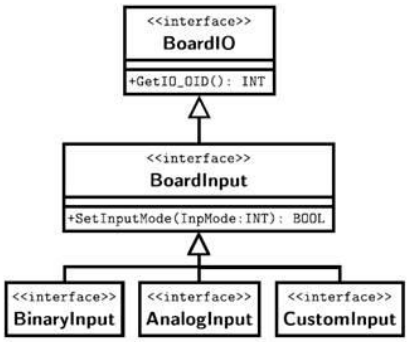
- IEC 61131-3: 2013

  Klasa, interfejs, dziedziczność, polimorfizm

  Klasa – POU: struktury danych, metody, dziedziczność pojedyncza

  Interfejs – implementacja klasy, dziedziczność wielokrotna

- Hierarchia interfejsu



- Rozszerzenia bloku funkcyjnego:

  Metody, interfejsy, dziedziczność

  Blok bez kodu = klasa

- Przykład

```
VAR_GLOBAL
  IOBAORD1 : RTDAC;
  IOBOARD2 : NIDAQ;
  GPS_SOURCE : GPS_NMEA;
END_VAR
```

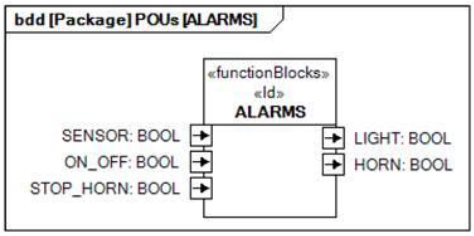
```
CLASS RTDAC IMPLEMENTS IOBinaryBoard, IOAnalogBoard
  ...
END_CLASS
```

Projektowanie oparte na modelu (MDD, MDE)

Model oprogramowania w formie diagramów graficznych przekształcanych do języka IEC-61131-3 lub do formatu PLCOpen.

- Profil UML zorientowany na sterowanie — DE, Fin, GR, PL
  DE: modAT4MS → CODESYS V3 (plug-in) — Munich (Aachen, Saarbrücken)
- SysML – bazuje na UML
  + rozszerzenia systemowe — DE, Fin, GR, PL
  DE: SyML-AT → CODESYS V3 — Munich (…)
- Standardy GRAFCET + GEMMA — ES
  ES: MeiA* →PLCOpen → SFC — Bilbao

- Blok funkcyjny w SysML

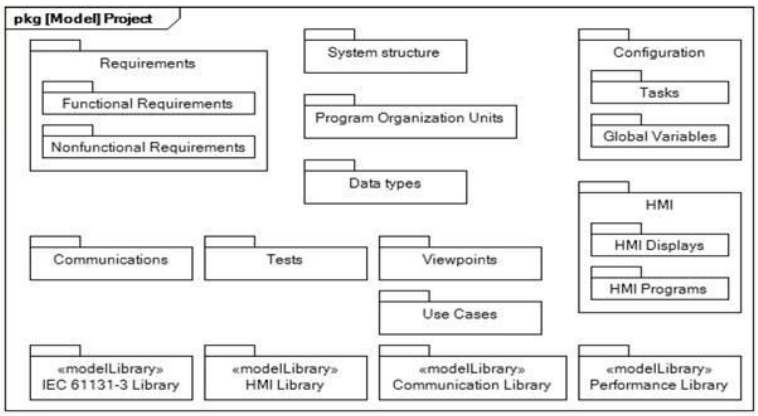


Szablon

```
FUNCTION_BLOCK ALARMS
VAR INPUT
...
END_INPUT
VAR_OUTPUT
...
END_OUTPUT
; (*empty*)
END_FUNCTION_BLOCK
```


SysML – charakterystyka

- SysML – zorientowany obiektowo + modelowanie strukturalne
  Rodzaje diagramów (9):
  wymagania: Requirements
  struktura: Package, Block Definition, Internal Block, Parametric
  zachowanie: Activity, Sequence, State Machine, Use Case
- Model ogólny jako Package Diagram (REQ, PKG, BDD, STM, IBD)

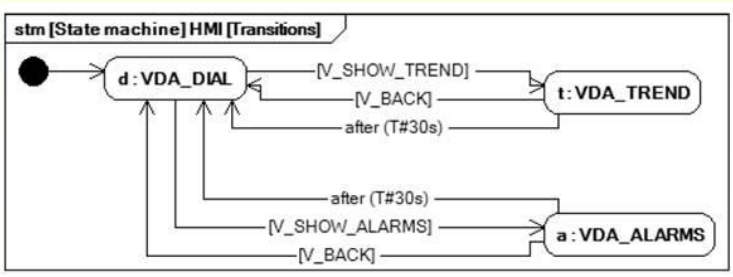


Model a kod

Automatyczna generacja kodu: szablony, kompletny kod.

- Przykład – graf przejść między ekranami HMI

STM



ST

```
PROGRAM HMI_TRANSITIONS
VAR_EXTERNAL
  V_CDISPLAY : INT;
  (* Global variables from STM *)
END_VAR
VAR
  _HMI_TRANSITIONS_STATE : INT := 0;
  _TIMER_AFTER : TON;
  _TIMER_AFTER_IN : BOOL;
  _TIMER_AFTER_PT : TIME;
END_VAR
_TIMER_AFTER(IN := _TIMER_AFTER_IN,
  PT := _TIMER_AFTER_PT);
CASE _HMI_TRANSITIONS_STATE OF
  DISPLAY_INDEX: IF (CONDITION) THEN
    _HMI_TRANSITIONS_STATE := NEXT_INDEX;
    (* Actions while leaving the state *)
    V_CDISPLAY := NEXT_INDEX;
    (* Actions while entering the state *)
  END_IF (...)
END_CASE (...)
END_PROGRAM
```


Język pośredni

- VMASM – zorientowany na IEC 61131-3

Funkcje

Mnemonic	Meaning	Operator
EXPT	Power	**
NEG	Negation	– (unary)
MUL	Multiplication	*
ADD	Addition	+ (arithm.)
CONCAT	String join	+ (text)
GT	Greater	>
AND	Logical and	&

Procedury

Mnemonic	Meaning
JMP	Unconditional jump
JNZ	Conditional jump
JR	Unconditional relative jump
CALB	Subroutine call
MEMCP	Copy memory block
MCD	Initialize data
GARD	Copy global memory to local area

- Składnia

```
[:label] instruction [operand1]
        [,operand2] [,operand3] ...
```

operand1 = rezultat (adres, wskaźnik)

- Przykład

```
MOTOR := (START OR MOTOR) AND
         NOT STOP AND NOT ALARM;
```

```
OR ?LR?ANDA005F, START, MOTOR
JZ ?LR?ANDA005F, :?PRJ?AND005E
NOT ?LR?ANDA0061, STOP
JZ ?LR?ANDA0061, :?PRJ?AND005E
MCD ?LR?ANDA005D, #01, #01
JMP :?PRJ?EAND0064
:?PRJ?AND005E
```

Kod binarny

```
09 20 03 00 00 00 08 00
1C 02 03 00 41 01
05 10 06 00 01 00
1C 02 06 00 41 01
1C 15 07 00 01 01
1C 00 47 01
```

Skalowalność i maszyna wirtualna VM

- Library Configuration File

  „Sprzęt” VM

```
<HARDWARE>
  <AddressSize>2</AddressSize>
  <MaxCodeAddress>0xFFFF</MaxCodeAddress>
  <MaxDataAddress>0xFFFF</MaxDataAddress>
</HARDWARE>
```

  Typy, funkcje, procedury, konwersje

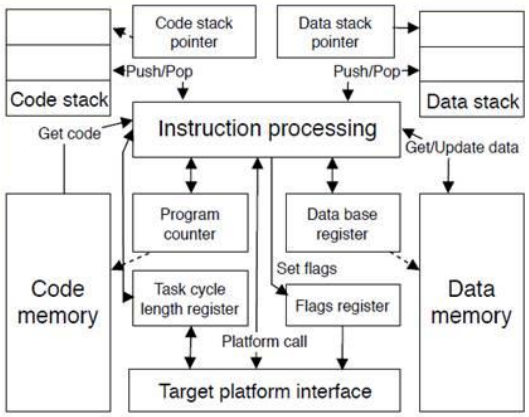
```
<TYPES>
  <deny-type name="LREAL" />
  <type name="USINT" implement="alias">
    <alias name="BYTE"/>
  </type>
</TYPES>
```

- Specyfikacja kompilatora i VM: *AddressSize, deny-type*

- Architektura VM

  Działanie:
  memory-to-memory

  CPU:
  AVR, ARM, x86



Model semantyczny I

Modele semantyczne są formalnym opisem języków programowania.

- Domeny semantyczne

$$
\begin{aligned}
BasicTypes &= OneByte + TwoBytes + \\
&+ FourBytes + EightBytes \\
Address &= \textbf{if } AddressSize = 2 \textbf{ then} \\
&TwoBytes \textbf{ else } FourBytes \\
Memory &= Address \to OneByte \\
CodeMemory &= Memory \\
Stack &= Address^* \\
CodeStack &= Stack \\
CodeReg &= Address \\
Flags &= TwoBytes
\end{aligned}
$$

- Stan

$$
\begin{aligned}
State = {} & DataMemory \times CodeMemory \times \\
& \times DataStack \times CodeStack \times \\
& \times DataReg \times CodeReg \times Flags
\end{aligned}
$$

Model semantyczny II

- Funkcje modelu

$$Get1BMem = (Address \times Memory) \rightarrow OneByte$$
$$Get2BMem = (Address \times Memory) \rightarrow TwoBytes$$

$$GetAddress = (Address \times Memory) \rightarrow Address$$

$$Push = (Stack \times Address) \rightarrow Stack$$
$$Pop = Stack \rightarrow (Address \times Stack)$$

- Interpretery wartości

$$BoolOf = OneByte \rightarrow BOOL$$
$$FromBool = BOOL \rightarrow OneByte$$

- Operatory (zakres), unifikacja (~ podstawienie), funkcja semantyczna

Rytro 2019 Tendencje *runtime* 21-22 maja

Denotacje a C

- Równanie denotacyjne

$$\mathcal{C}[\![command]\!] = State \rightarrow State$$

C – funkcja semantyczna

- Denotacja skoku JMP

$$\mathcal{C}[\![\texttt{JMP:clbl}]\!] = \lambda s.$$
$$(cm, dm, cs, ds, cr, dr, flg) := s$$
$$clbl := GetAddress(cr, cm)$$
$$s_1 := (cm, dm, cs, ds, clbl, dr, flg)$$
$$s_1$$

funkcja λ

- Odpowiednik C

```
case 0x00: /* JMP unconditional jump */
{
 ADDRESS clbl = GetCodeAddress();
 codeReg = clbl;
}
break;
```

Implementacja „zrób-to-sam" I

- Zasada

  Urządzenia producentów SMES, takie jak dedykowane sterowniki, regulatory procesowe, stacje I/O, itp. są wyposażone w następujące oprogramowanie podstawowe:

  – niskopoziomowe procedury obsługi sprzętu i procesowych I/O

  – *drivery* komunikacyjne określonych protokołów

  Celem implementacji środowiska CPDev (*framework*) jest wtedy:

  1) zintegrowanie procedur niskopoziomowych w maszynie wirtualnej
  2) opracowanie konfiguratora wiążącego zmienne projektu z portami *driverów* i I/O.

- Prototypy funkcji niskopoziomowych maszyny VM (*target platform interface*)

  VMP_LoadConfiguration
  VMP_PreRunConfiguration
  VMP_PreCycle
  VMP_PostCycle
  VMP_CurrentTime
  VMP_ReadRTC
  VMP_PostRunConfiguration

  W prototypach (początkowo pustych) inżynier implementujący osadza kod C/C++ procedur niskopoziomowych.

Rytro 2019 Tendencje *runtime* 21-22 maja

Implementacja „zrób-to-sam" II

- Kompilator generuje plik XML z danymi zmiennych globalnych projektu:

  LName – nazwa lokalna
  PName – nazwa fizyczna
  Addr – heksadecymalny
  AddrType – globalny, względny
  Size – 1, 2, 4, 8 bajtów
  Type – typ IEC 61131-3
  PType – typ fizyczny
  VarFlags – flagi zmiennej

```
<VAR LName="START" PName="START_STOP.START"
     Addr="0000" AddrType="gdlabel" Size="1"
     Type="BOOL" PType="$DEFAULT.BOOL"
     VarFlags="00004008" />
```

- Inżynier opracowuje konfigurator wiążący zmienne globalne z portami *driverów* komunikacyjnych i I/O. Tabela powiązań jest przesyłana do sterownika wraz z kodem binarnym projektu.

- Mini–DCS

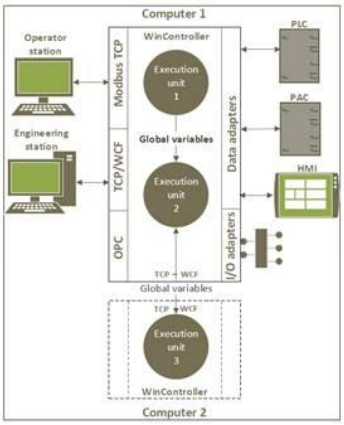
Communication task table

No	Slave no.	Function	Remote addr.	Data number	Name	Conversion
1	1	FC3-read reg.	4003	3	START	16->8 bit
2	2	FC16-write reg.	4205	2	MOTOR	8->16 bit

Names

Skomplikowany *software* I

- System DCS z komputerami PC, które współpracując ze sterownikami PLC/PAC wykonują w czasie rzeczywistym skomplikowane oprogramowanie wspierające (zastosowania SMES)
  Koncepcja – implementacja kilku powiązanych maszyn wirtualnych VM na wspólnej platformie.
- WinController *runtime*

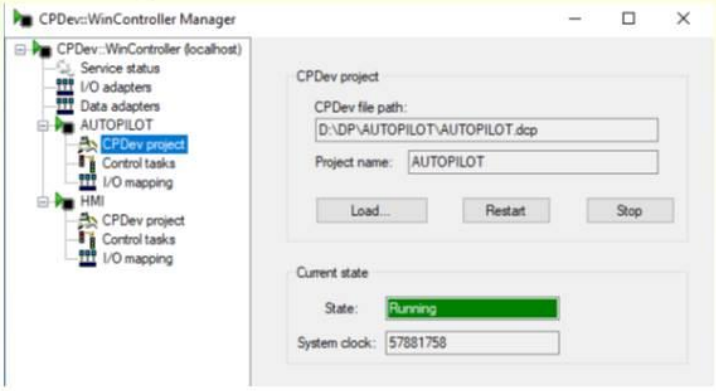


VM = *execution unit* (~ PLC)
Wymiana danych – zmienne globalne
Data adapters – PLC, PAC, HMI
I/O adapters – karty I/O

Rytro 2019 Tendencje *runtime* 21-22 maja

Skomplikowany *software* II

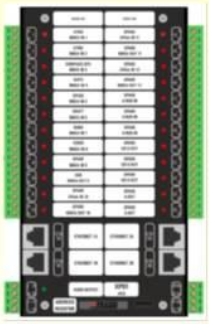
- WinController jako usługa *Windows service* funkcjonuje bez przerwy w tle (*background*). Jest uruchamiany automatycznie w momencie włączenia komputera.
- WinController Manager



I/O mapping: łączy jednostki wykonawcze, Wincontrollery, Data / I/O adapters

Zastosowania

- Praxis A.I., NL - statki



Ethernet, CAN
NMEA, OPC

- iGrid T&D, ES – energia

IEC 61850
IEC 60870
Modbus RTU/TCP
Profibus DP

- NiT, PL – transport, obiekty komunalne

GSM/GPRS

- Bart-Com, PL – inteligentny dom, przemysł

Profibus DP, Modbus RTU

Rytro 2019 Zastosowania i uzupełnienia 21-22 maja

IoT. FPGA

- IoT
  Inteligentne czujniki: farmy wiatrowe, pompy, pojazdy, domy, …
  Software: *made-to-measure PLC* – mały *footprint* (skalowalny)
  MQTT, OPC UA: *machine-to-machine*, serwery w chmurze.
- FPGA
  Początkowo: tekstowy IL → FPGA
  Aktualnie: LD, SFC → IL → FPGA
  Złożone: n ×VM → wielordzeniowe FPGA (~ WinController)
- CPS – *Cyber Physical System*
  Cyber (IT): informacja, przetwarzanie, komunikacja
  Physical: obiekty – przemysł, transport, medycyna, AGD, …
  System CPS powstaje poprzez integrację informatycznych i fizycznych elementów i/lub podsystemów CPS, który dla otoczenia wygląda jak samodzielna jednostka.

Wielordzeniowe CPU. Komunikacja. Testy

- Wielordzeniowe CPU
  Aktualnie: program podzielony na sekcje, dedykowane biblioteki
  Proponowane: rdzeń = *execution unit* (PLC), systemowa wymiana danych, alternatywa dla wielozadaniowości
- Komunikacja
  Komunikacja przemysłowa – deterministyczna, jednorodna (cykl)
  Sieci publiczne – niedeterministyczne, niejednorodne (heterogeniczne, Internet, GSM/GPRS), protokoły MQTT, OPC UA
  Implementacja, diagnostyka, przyszłość = ?
- Testy
  Testy jednostkowe – POU (IEC 61131-3, IEC 61499)
  Asercyjne bloki funkcyjne, testy tabelowe
  Dedykowany język zawierający asercje.