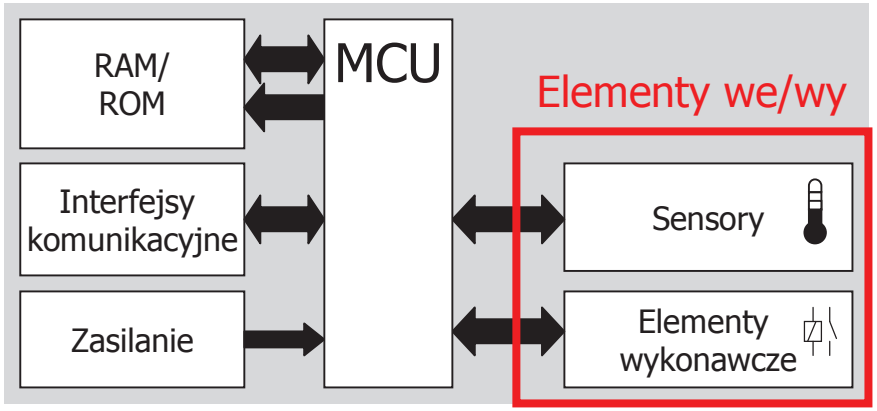
Maszynowa generacja oprogramowania dla dwumikrokontrolerowych węzłów Internetu Rzeczy

Aleksander Pruszkowski
Politechnika Warszawska, Instytut Telekomunikacji
00-665 Warszawa, ul. Nowowiejska 15/19
apruszko@tele.pw.edu.pl

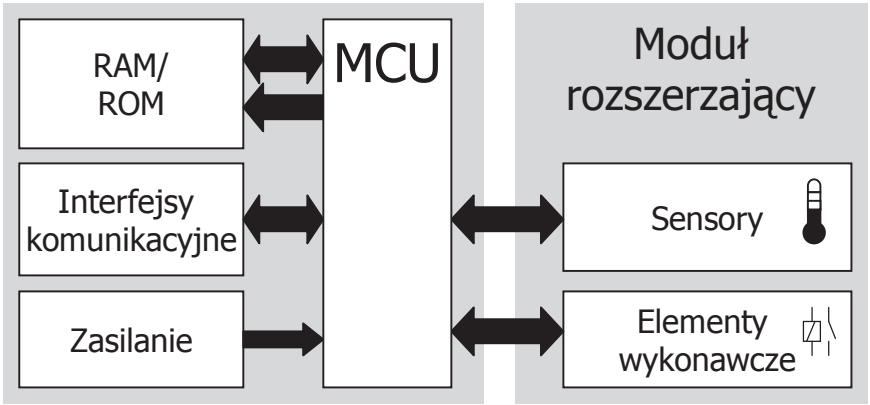
Wstęp

- Typowy węzeł Internetu Rzeczy - wybrane problemy, cd.
  - Zmiana elementów we/wy (np.: na życzenie klienta) powoduje znaczny wzrost kosztów wytworzenia nowej wersji urządzenia



Wstęp

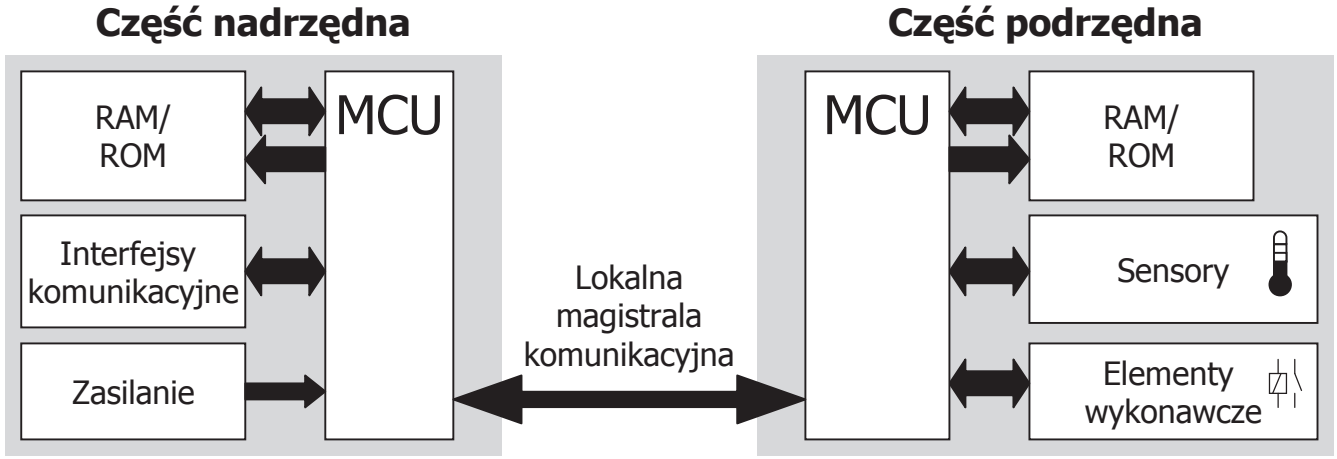
- Budowa modularna węzła Internetu Rzeczy



  - Podejście znane od dawna (IBM PC)
  - Mniej powszechne w WSN i węzłach Internetu Rzeczy
  - Wyzwania
    - zbyt prosta budowa modułu rozszerzającego utrudnia realizację skomplikowanych zadań
      - np.: sterowanie czasowo krytyczne matrycowym wyświetlaczem LED, wbudowanym w moduł rozszerzający
    - ograniczona liczba sygnałów łączących obie części węzła

Wstęp

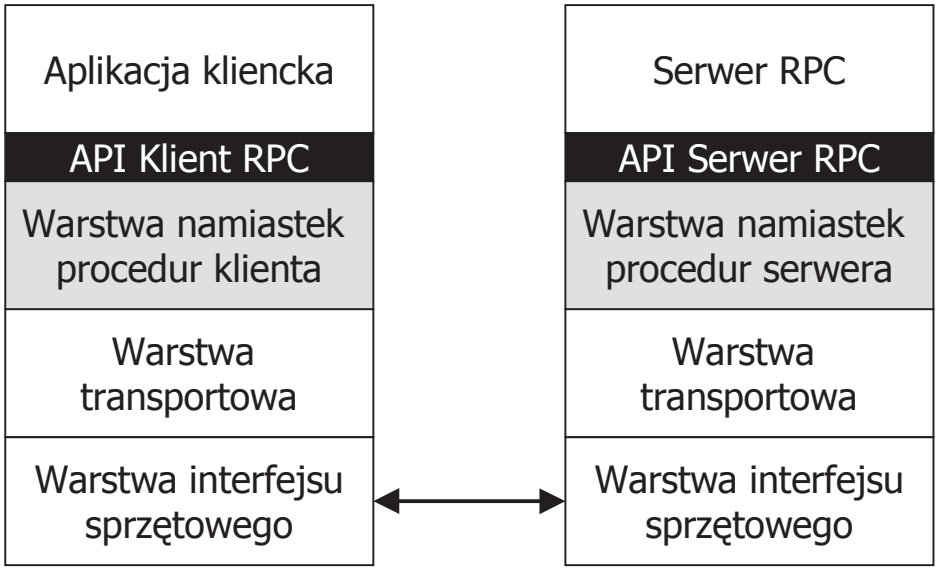
- Co jest proponowane?
  - Dodanie drugiego mikrokontrolera w module rozszerzającym
    - obie części połączy niemal niezawodna i prosta lokalna magistrala komunikacyjna punkt-punkt (np.: I2C, UART, ...)



  - Zastosowanie technik zdalnego wywoływania procedur (RPC) do utworzenia modularnego węzła Internetu Rzeczy
  - Automatyczne generowanie kodu implementacji protokołu zdalnego wywoływania procedur
    - minimalizacja wielkości implementacji i wielkości przesyłanych danych

Podobne podejścia

- Zdalne wywoływanie procedur (RPC)
  - Podejście typowe dla łączności IP - zbyt ciężkie dla węzłów modularnych połączonych lokalną magistralą komunikacyjną



Propozycja rozwiązania - protokół nanoRPC

- Automatyczne generowanie implementacji protokołu - etapy
  1) Wizja projektanta systemu -> utworzenie opisu (zbliżonego do C)
  2) Opis usług XML - czytelny dla generatora zapis wizji
  3) Właściwe wygenerowanie kodu protokołu nanoRPC

Opis działania usług z podaniem API w C:

Ustawienie alarmu
uint8_t setAlarm(uint32_t a1, uint8_t a2);
a1 - czas nowego alarmu
a2 - typ generowanego alarmu
zwraca: faktyczny stan alarmu
(„przeciążenie” przez użytkownika)

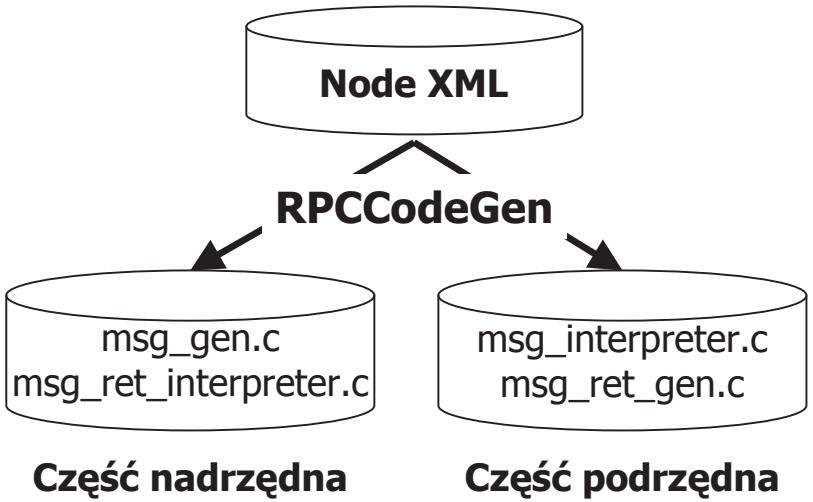
Czas alarmu
uint32_t getAlarm(void);
zwraca: czas najbliższego alarmu

Opis usług w formacie XML:

```
<service>
 <def>setAlarm</def>
  <args>
    <type>uint32_t</type><type>uint8_t</type>
  </args>
  <result>uint8_t</result>
</service>
<service>
  <def>getAlarm</def>
  <result>uint32_t</result>
</service>
```

Propozycja rozwiązania - protokół nanoRPC

- Automatyczne generowanie implementacji protokołu, cd.
  - Narzędzie - RPCCodeGen
    - Python oraz biblioteka obsługi plików XML
    - Generacja modułów w C dla części nadrzędnej i podrzędnej węzła
      - wstępna konwersja typów C na typy o ustalonym formacie
        - np.: unsigned char -> uint8_t, unsigned long -> uint32_t



Propozycja rozwiązania - protokół nanoRPC

- Próbka działania narzędzia - wygenerowana implementacja
  - Generacja identyfikatorów usług

msg_gen.c – generator wywołań (część nadrzędna)

```
#define CHRPC              0x41
#define CONST_SETALARM     2

res_t setAlarm(uint32_t a,uint8_t b){
  uint8_t setAlarmMsg[6]={CONST_SETALARM};
  uint8_t p=1;
  uint32_t t_a=MHTONL(a);
  uint8_t t_b=b;
  memcpy(&(setAlarmMsg[p]),&t_a,
              sizeof(uint32_t));
  p+=sizeof(uint32_t);
  memcpy(&(setAlarmMsg[p]),&t_b,
               sizeof(uint8_t));
  p+=sizeof(uint8_t);
  return sendPoUART(CHRPC,p,setAlarmMsg);
}
...
```

msg_interpreter.c – interpreter wiadomości (część podrzędna)

```
#define CHRPC              0x41
#define CONST_SETALARM     2
...

void recvPoUART(uint8_t c, uint8_t l, uint8_t *r){
  if(c==CHRPC){
    switch(r[0]){
      case CONST_SETALARM:{
        uint8_t p=1;
        uint32_t a;
        uint8_t b;
        if(l!=6)
           ERROR(CONST_SETALARM);
        memcpy(&a, &(r[p]), sizeof(uint32_t));
        p+=sizeof(uint32_t);
        memcpy(&b, &(r[p]), sizeof(uint8_t));
        setAlarmSrv(MNTOHL(a),b);
        break;
      }
...}}}
```

Propozycja rozwiązania - protokół nanoRPC

- Próbka działania narzędzia - wygenerowana implementacja
  - Ustalanie kolejności bitów przekazywanych argumentów

msg_gen.c – generator wywołań (część nadrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2

res_t setAlarm(uint32_t a,uint8_t b){
  uint8_t setAlarmMsg[6]={CONST_SETALARM};
  uint8_t p=1;
  uint32_t t_a=MHTONL(a);
  uint8_t t_b=b;
  memcpy(&(setAlarmMsg[p]),&t_a,
              sizeof(uint32_t));
  p+=sizeof(uint32_t);
  memcpy(&(setAlarmMsg[p]),&t_b,
               sizeof(uint8_t));
  p+=sizeof(uint8_t);
  return sendPoUART(CHRPC,p,setAlarmMsg);
}
...
```

msg_interpreter.c – interpreter wiadomości (część podrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2
...

void recvPoUART(uint8_t c, uint8_t l, uint8_t *r){
  if(c==CHRPC){
    switch(r[0]){
      case CONST_SETALARM:{
        uint8_t p=1;
        uint32_t a;
        uint8_t b;
        if(l!=6)
           ERROR(CONST_SETALARM);
        memcpy(&a, &(r[p]), sizeof(uint32_t));
        p+=sizeof(uint32_t);
        memcpy(&b, &(r[p]), sizeof(uint8_t));
        setAlarmSrv(MNTOHL(a),b);
        break;
      }
...}}}
```

Propozycja rozwiązania - protokół nanoRPC

- Próbka działania narzędzia - wygenerowana implementacja
  - Precyzyjna kontrola rozmieszczania argumentów w treści wiadomości

msg_gen.c – generator wywołań (część nadrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2

res_t setAlarm(uint32_t a,uint8_t b){
  uint8_t setAlarmMsg[6]={CONST_SETALARM};
  uint8_t p=1;
  uint32_t t_a=MHTONL(a);
  uint8_t t_b=b;
  memcpy(&(setAlarmMsg[p]),&t_a,
              sizeof(uint32_t));
  p+=sizeof(uint32_t);
  memcpy(&(setAlarmMsg[p]),&t_b,
               sizeof(uint8_t));
  p+=sizeof(uint8_t);
  return sendPoUART(CHRPC,p,setAlarmMsg);
}
...
```

msg_interpreter.c – interpreter wiadomości (część podrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2
...

void recvPoUART(uint8_t c, uint8_t l, uint8_t *r){
  if(c==CHRPC){
    switch(r[0]){
      case CONST_SETALARM:{
        uint8_t p=1;
        uint32_t a;
        uint8_t b;
        if(l!=6)
           ERROR(CONST_SETALARM);
        memcpy(&a, &(r[p]), sizeof(uint32_t));
        p+=sizeof(uint32_t);
        memcpy(&b, &(r[p]), sizeof(uint8_t));
        setAlarmSrv(MNTOHL(a),b);
        break;
      }
...}}}
```

Propozycja rozwiązania - protokół nanoRPC

- Próbka działania narzędzia - wygenerowana implementacja
  - Tworzenie implementacji odpowiadających sobie namiastek

msg_gen.c – generator wywołań (część nadrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2

res_t setAlarm(uint32_t a,uint8_t b){
  uint8_t setAlarmMsg[6]={CONST_SETALARM};
  uint8_t p=1;
  uint32_t t_a=MHTONL(a);
  uint8_t t_b=b;
  memcpy(&(setAlarmMsg[p]),&t_a,
              sizeof(uint32_t));
  p+=sizeof(uint32_t);
  memcpy(&(setAlarmMsg[p]),&t_b,
               sizeof(uint8_t));
  p+=sizeof(uint8_t);
  return sendPoUART(CHRPC,p,setAlarmMsg);
}
...
```

msg_interpreter.c – interpreter wiadomości (część podrzędna)

```
#define CHRPC            0x41
#define CONST_SETALARM   2
...

void recvPoUART(uint8_t c, uint8_t l, uint8_t *r){
  if(c==CHRPC){
    switch(r[0]){
      case CONST_SETALARM:{
        uint8_t p=1;
        uint32_t a;
        uint8_t b;
        if(l!=6)
           ERROR(CONST_SETALARM);
        memcpy(&a, &(r[p]), sizeof(uint32_t));
        p+=sizeof(uint32_t);
        memcpy(&b, &(r[p]), sizeof(uint8_t));
        setAlarmSrv(MNTOHL(a),b);
        break;
      }
...}}}
```

Propozycja rozwiązania - protokół nanoRPC

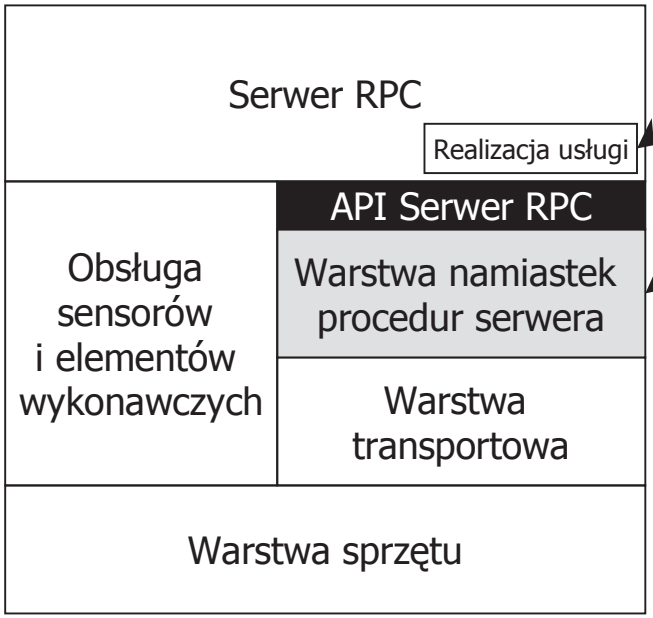
- Współdziałanie implementacji protokołu z resztą oprogramowania
  - Komunikacja - wiadomości „PoUART”

Channel ID	Len	Payload	CRC-8

  - nanoRPC jest wspierane przez oprogramowanie zainstalowane w obu częściach węzła
    - funkcje dostarczania niezawodnego pakietów do drugiej części węzła
    - funkcje ustalania kolejności końcówek (MHTONL, MNTOHL)
    - obsługi błędów (ERROR)

Propozycja rozwiązania - protokół nanoRPC

- Współdziałanie implementacji protokołu z resztą oprogramowania, cd.
  - Realizacja zdalnych usług w części podrzędnej - przykład
    - kod tworzony „ręcznie” - specyficzny dla usługi realizowanej na określonym sprzęcie




```
void setAlarmSrv(uint32_t a, uint8_t b){
  ...
  setAlarmRet(alarmState);
}
void recvPoUART(uint8_t c, uint8_t l, uint8_t *r){
  if(c==CHRPC){
    switch(r[0]){
      case CONST_SETALARM:{
        ...
        setAlarmSrv(MNTOHL(a),b);
  ...
}
```

```
...
int main(){
  initHW();initCommunication();
  for(;;){
    ...
    communicationProcess();
  }
}
```