UPPAAL

Grzegorz Olędzki

prezentacja w ramach
seminarium Protokoły komunikacyjne

grudzień 2004

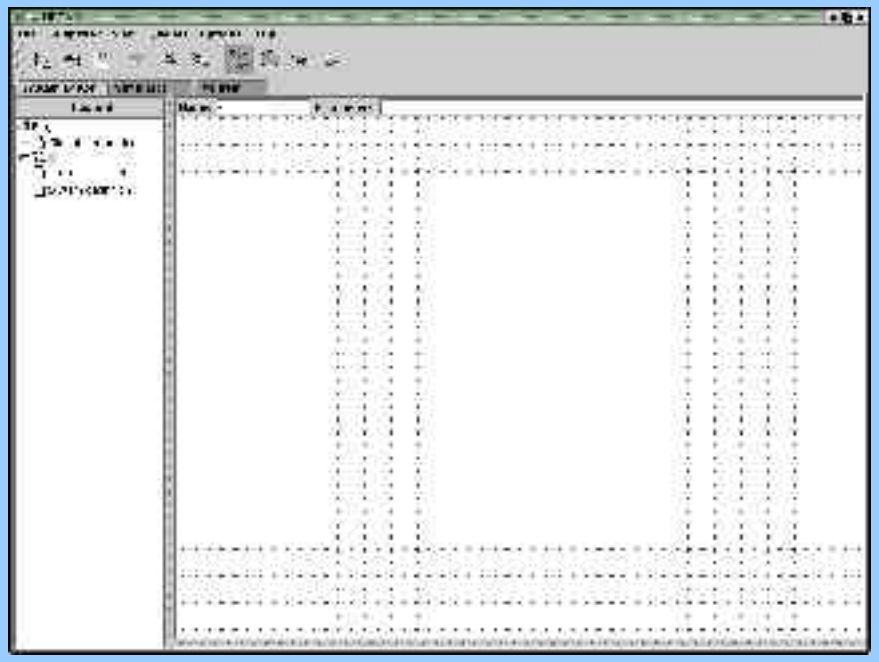
Co to jest?

*"**Uppaal** is an integrated tool environment for modeling, validation and verification of real-time systems modeled as networks of **timed automata**, extended with data types (bounded integers, arrays, etc.)."*

uppaal.com

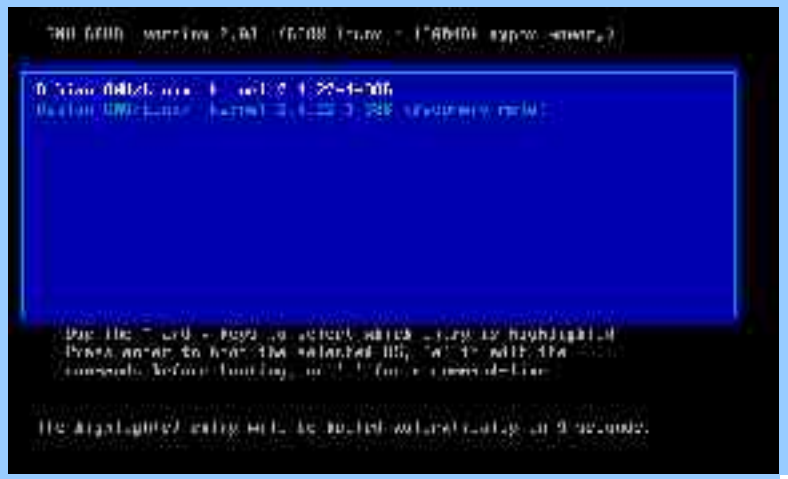
UPPAAL – jak zacząć

- (wymagana jest Java)
- ściągnąć z uppaal.com najnowszą wersję (trzeba podać dane) – będzie to plik uppaal.zip
- rozpakować zipa
- uruchomić skrypt uppaal
- działa :)

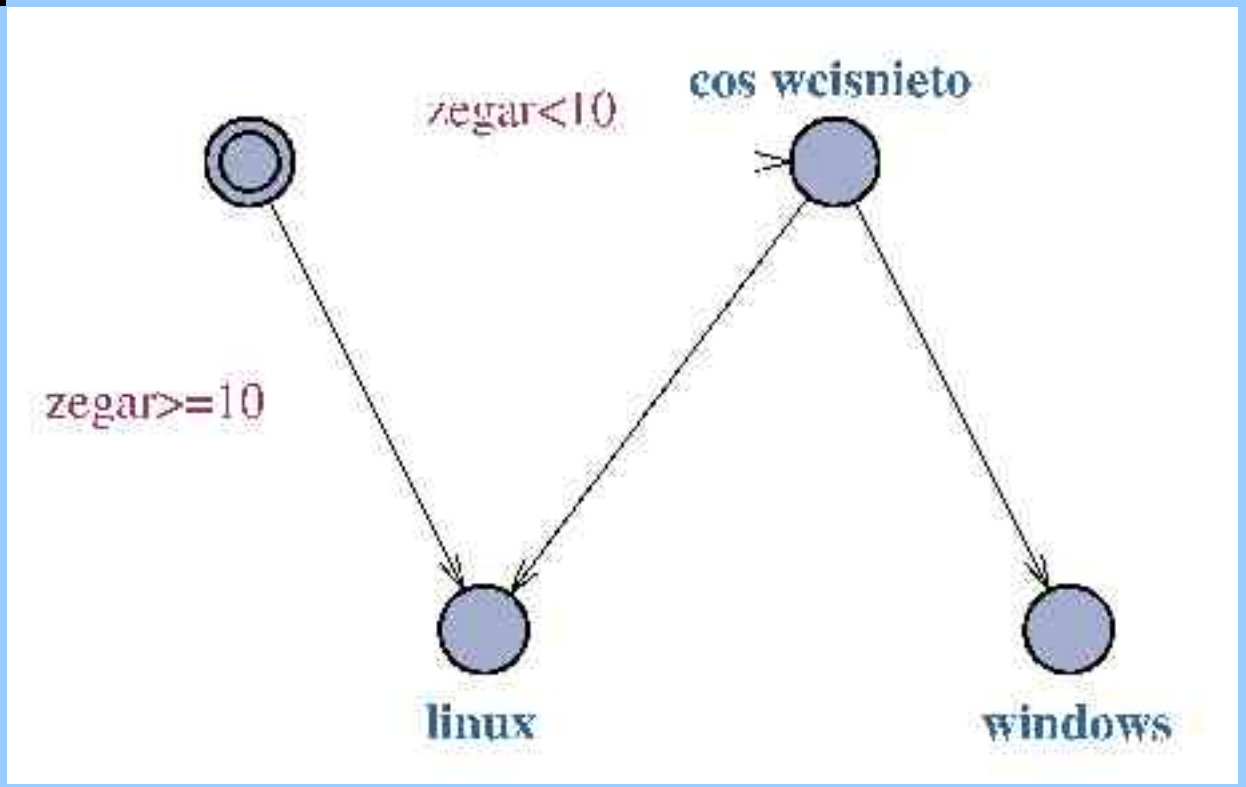


Automaty z czasem

Przykład 1: “Boot loader”



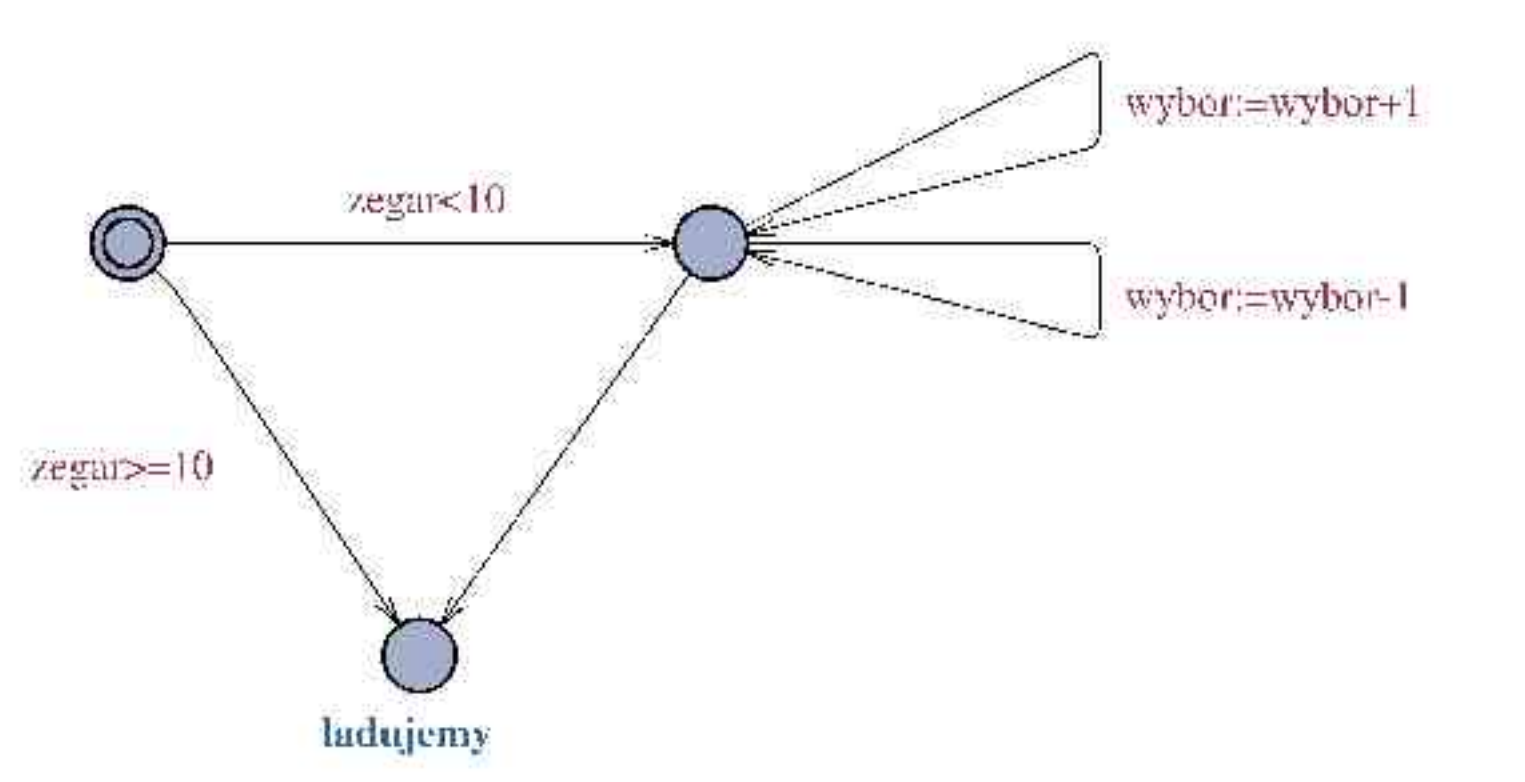
- definicja zegara
  `clock zegar;`
- warunki na krawędziach (guards)
  `zegar<10`
  `zegar>=10`



“Automaty ze zmiennymi”

Oprócz czasu dodajemy też zmienne

Przykład 2: “Boot loader” - wersja uogólniona



- definicja zmiennej (całkowitej)

```
int wybor:=0;
```

- akcje na krawędziach (updates)

```
wybor:=wybor-1
wybor:=wybor+1
```

Bootloader w UPPAALu

- Modelujemy automat (“rysujemy” stany, krawędzie, wpisujemy warunki i akcje).
- Definiujemy zegar i zmienną.
- Przeprowadzamy symulację (zakładka Simulator).

UPPAAL - weryfikacja

Oprócz modelowania i śledzenia dostępna jest możliwość automatycznej weryfikacji.

Można zadawać pytania, czy:

- **`E<>p`** – istnieje ścieżka, na której p jest choć raz spełnione,
- **`A[]p`** – na wszystkich ścieżkach p jest zawsze spełnione,
- **`E[]p`** – istnieje ściezka, na której p jest zawsze spełnione,
- **`A<>p`** – na wszystkich ścieżkach p jest choć raz spełnione,
- **`p-->q`** – za każdym razem jeśli p jest spełnione, to q też.

UPPAAL - weryfikacja

- Zapytania można budować w prosty sposób przy użyciu typowych operatorów.
- Proces1 jest w stanie A: `Proces1.stanA`
- Parę przykładowych zapytań:
  - `E<> Bootloader.ladujemy && (Bootloader.wybor==3)`
    = czy da się załadować system o numerze 3?
  - `A[] Bootloader.wybor>=0`
    = czy zawsze aktualny wybór jest nieujemny?
  - itd...

UPPAAL - weryfikacja

- Oprócz odpowiedzi na pytanie (TAK/NIE) weryfikator potrafi podać ścieżkę, która stanowi dowód zapytania (Options| Diagnostic Trace).
- Dotyczy to zapytań, które można udowodnić przez pokazanie palcem przykładu, czyli:
  - **`E<>p`** (istnieje ... choć raz ...) przy odp. TAK
  - **`A[]p`** (na każdej ... zawsze ...) przy odp. NIE

Więcej automatów

- Do tej pory rozważaliśmy naraz tylko jeden automat.
- A w odpowiedzi na pytanie “co to jest Uppaal?” była mowa o “***networks** of timed automata*”.
- Rozważmy więc sieć równoległych automatów.

Komunikacja przez kanały

- Symetryczna komunikacja przez tzw. kanały:
  - deklarujemy kanał:
    `chan nazwaKanalu;`
  - krawędzie, na których będziemy dokonywać synchronizacji etykietujemy w obu procesach:
    `nazwaKanalu?`
    `nazwaKanalu!`

Kanały - przykład

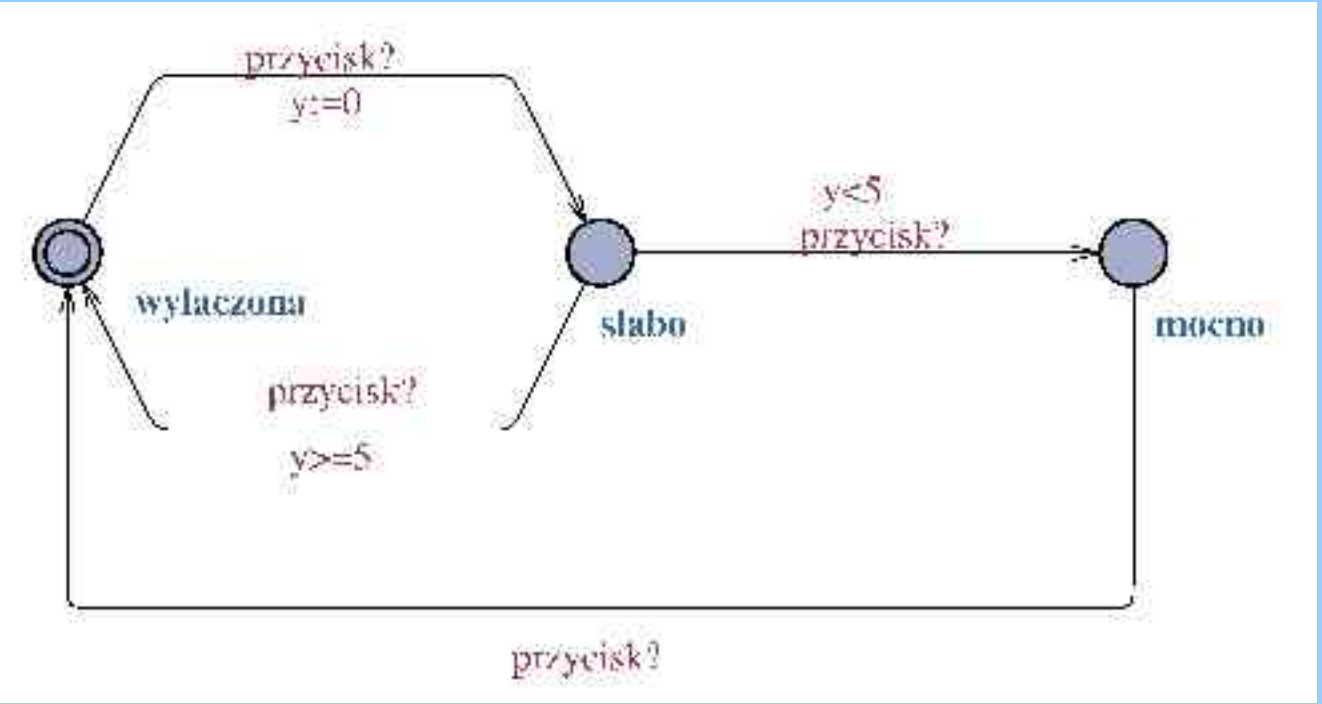
Przykład 3: “Lampa”

Modelujemy zachowanie lampy z jednym przyciskiem, który:

- jeśli wciśniemy raz, to lampa zapala się słabym światłem,
- jeśli wciśniemy szybko dwa razy, to lampa zaświeci się mocno.

Człowiek

Lampa



Algorytm Petersona

proces A

```
chce[A]:=true;

kolej:=B;

while
 (kolej==B && chce[B]);

// sekcja krytyczna

chce[A]:=false;
```

proces B

```
chce[B]:=true;

kolej:=A;

while
 (kolej==A && chce[A]);

// sekcja krytyczna

chce[B]:=false;
```

Algorytm Petersona

```
chce[JA]:=true;

kolej:=NIE_JA;

while
 (kolej!=JA && chce[NIE_JA]);

// sekcja krytyczna

chce[JA]:=false;
```

stworzymy dwie instancje takiego procesu:

JA:=A NIEJA:=B	JA:=B NIEJA:=A

Algorytm Petersona

nic:

```
chce[JA]:=true;
```

chcę:

```
kolej:=NIE_JA;
```

czekam:

```
while
 (kolej!=JA && chce[NIE_JA]);
```

sekcja krytyczna:

```
// sekcja krytyczna
```

```
chce[JA]:=false;
```

Algorytm Petersona

- Zamodelujemy szablon procesu
- Stworzymy jego dwie instancje
- Prześledzimy, jak się zachowują
- Zapytamy weryfikatora, czy system jest bezpieczny

Niezmiennik stanu

- Warunek, który stan zobowiązuje się spełniać, np.:
  - `zegar<5` – wyjście ze stanu nastąpi zanim zegar dojdzie do 5,
- Warunki są ograniczone, w szczególności nie możemy:
  - żądać, żeby wartość zegara była większa od czegoś

Atrybuty stanów

- początkowy (*initial*) - od niego rozpoczyna się przejście po automacie,
- pilny (*urgent*) – podczas przebywania w danym stanie – czas nie może płynąć.
- związany (*committed*) – po wejściu do takiego stanu, w następnym kroku, musimy wyjść.

Przykład z życia

- Firma Bang&Olufsen, “znany” producent sprzętu audio-video, opracowała protokół do komunikacji między urządzeniami.
- Rozważamy kanał, do którego równoczesny dostęp ma kilka urządzeń.
- Każde z nich potrafi nadawać sygnał jak i odbierać co inni wysyłają.

Bang&Olufsen

- Informacje dzielone są na ramki. Każda z nich jest postaci: $\text{frame} ::= T_5\{T_1|T_2|T_3\}^{\geq 15}T_4$
  - symbol T5 – początek ramki
  - co najmniej 15 symboli ze zbioru {T1,T2,T3}
  - symbol końcowy T4 – koniec ramki

Szczegóły techniczne

- Sygnał Ti to napięcie 5V przez 2*1562*i mikrosekund, przed i po 1562 mikrosekund napięcia 0V.

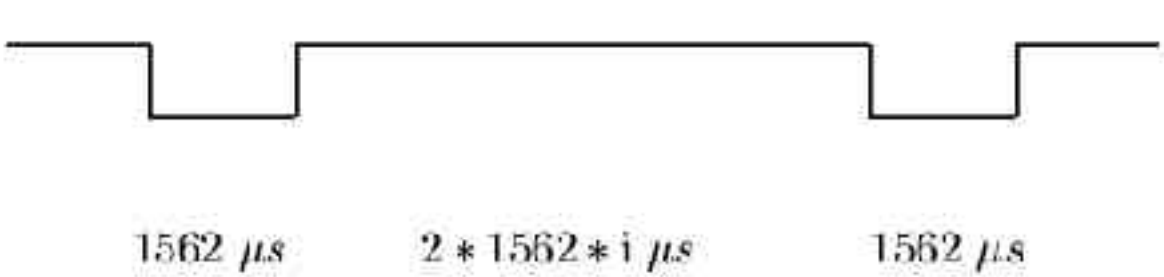
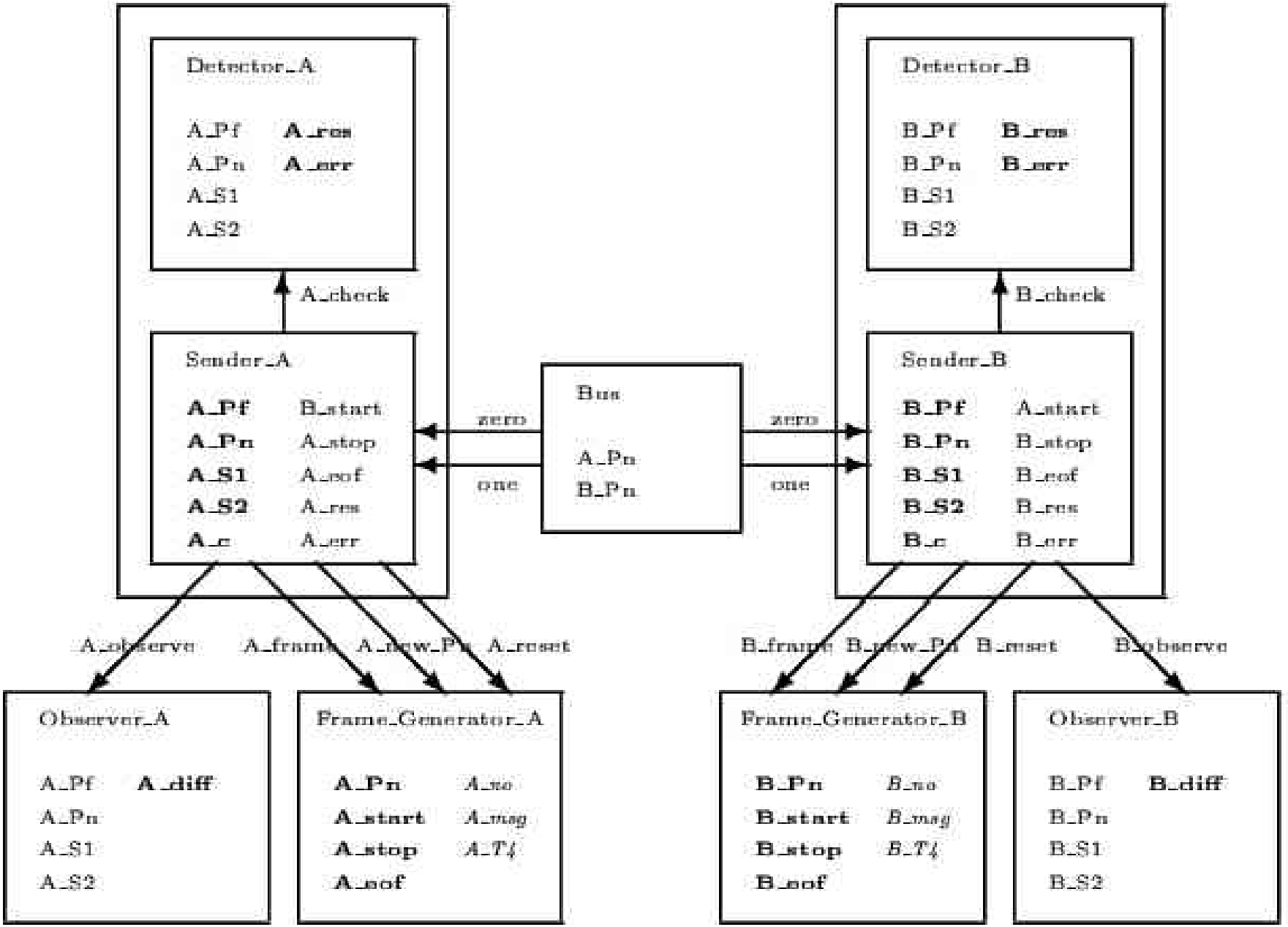


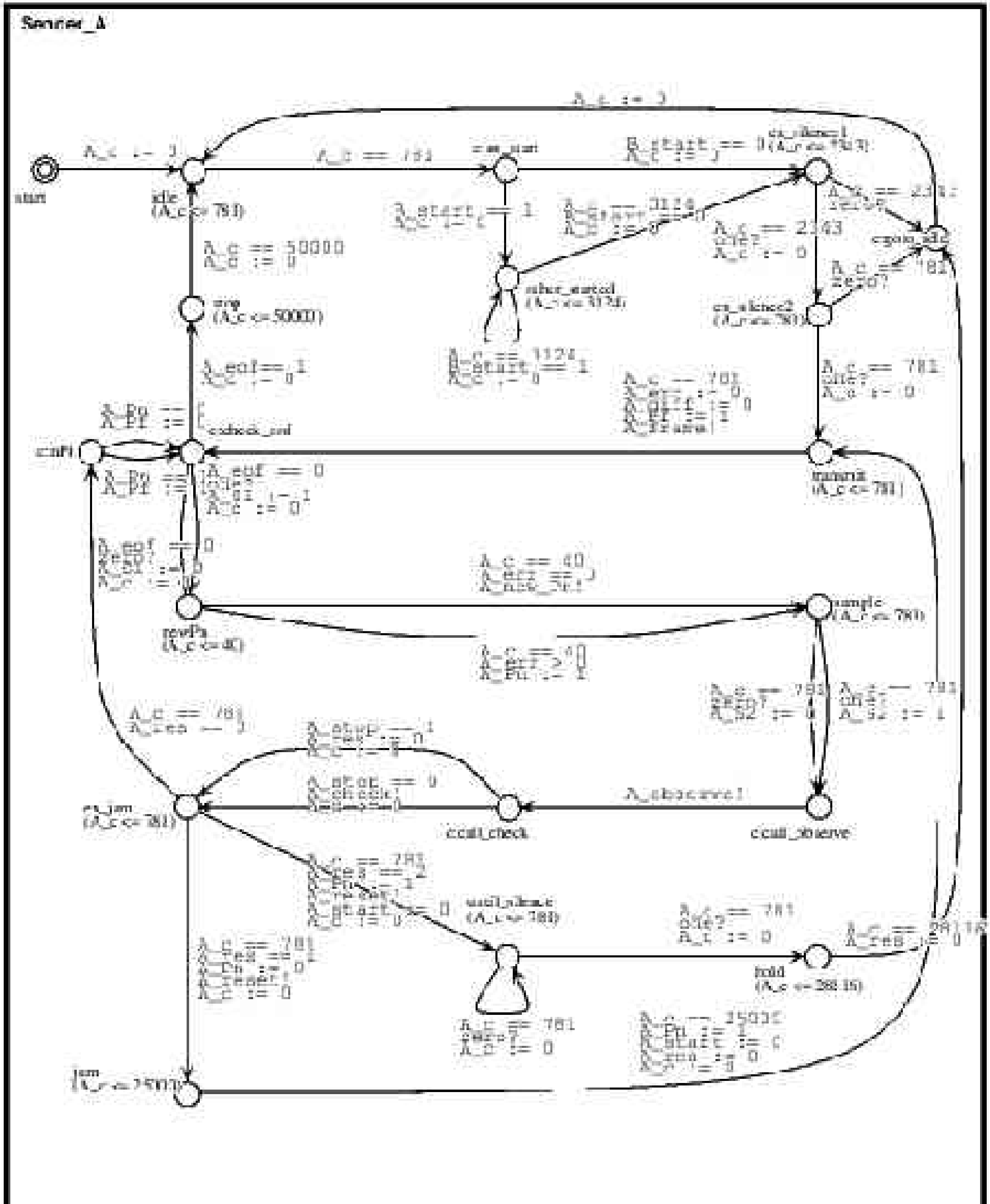
Fig. 3. Physical representation of a T_i message.

Kanał ma taką specyfikę, że zero ma pierwszeństwo – tj. kanał działa jak funkcja logiczna **i** na wejściach ze wszystkich nadajników.

Przedziały czasowe i kolizje

- Ściśle zdefiniowany jest proces nadawania:
  - faza inicjalizacji
  - nadawanie (ze sprawdzaniem czy to, co jest w eterze, to jest to, co nadawaliśmy)
  - w przypadku błędu – obsługujemy powstałą kolizję


Detector_A
A_Pf
A_Pn
A_S1
A_S2
A_res
A_err
A_check
Sender_A
A_Pf
A_Pn
A_S1
A_S2
A_c
B_start
A_stop
A_eof
A_res
A_err
zero
one
Bus
A_Pn
B_Pn
zero
one
Detector_B
B_Pf
B_Pn
B_S1
B_S2
B_res
B_err
B_check
Sender_B
B_Pf
B_Pn
B_S1
B_S2
B_c
A_start
B_stop
B_eof
B_res
B_err
A_observe
A_frame
A_new_Pn
A_reset
B_frame
B_new_Pn
B_reset
B_observe
Observer_A
A_Pf
A_Pn
A_S1
A_S2
A_diff
Frame_Generator_A
A_Pn
A_start
A_stop
A_eof
A_no
A_msg
A_Tf
Frame_Generator_B
B_Pn
B_start
B_stop
B_eof
B_no
B_msg
B_Tf
Observer_B
B_Pf
B_Pn
B_S1
B_S2
B_diff


Sender_A
start
idle
A_c == 50000
A_c := 0
transmit
sample
ccall_check
ccall_observe
A_observe!
hold
jam

B&O - poprawność

- Zrobiono model do protokołu, bo ten w rzadko występujących sytuacjach działał nieprawidłowo.
- Powstało pytanie – co to znaczy prawidłowo?
  - (1) jeśli ramka wysłana przez nadawcę X zostanie zniszczona, to X ma to zauważyć,
  - (2) jeśli jeden z nadawców zauważa kolizję, to wszyscy powinni ją zauważyć.

Rozwiązanie

- Uproszczono protokół (m.in. przez ograniczenie, że tylko jeden znaczący symbol w ramce jest przesyłany).
- Wprowadzono go do UPPAALa.
- Ten wykrył błąd (czas obliczeń: 6.27 minuty, pamięć potrzebna: 32MB).
- Poprawiono protokół.
- Udowodniono jego poprawność (czas obliczeń: (czas: 30 minut, pamięć: 90MB).
- Działa.

Bibliografia

- Strona domowa projektu: www.uppaal.com.
- Pomoc dostępna w programie (F1).
- Tutoriale:
  - “mały”: http://www.it.uu.se/research/group/darts/uppaal/tutorial.ps.gz
  - “duży”: http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf
- Przykłady załączone do programu.
- Praca o Bang&Olufsenie:
  http://www.it.uu.se/research/group/darts/papers/texts/hsll-unpublished97.ps.gz

Czy są jakieś pytania?

To wszystko.
Dziękuję.