

Modelling CAN Communication for Intra-Vehicular Applications

Vancouver, 28 April 2011

Dr. Dario Kresic, University of Zagreb

Structure

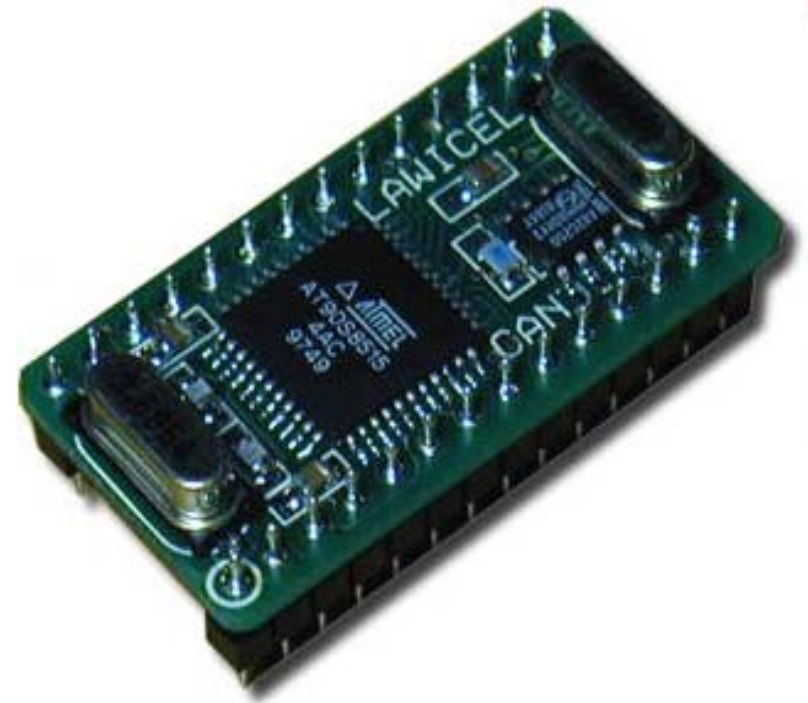
- Controller Area Network (CAN)
- Medium access in CAN
- Modelling time
- Our CAN model
- Checking CAN properties
- Delay bounds

Controller Area Network (CAN)

- intra-vehicular communication
- 60-100 electronic control units (ECU)
- **CAN bus**
- reliability, real-time

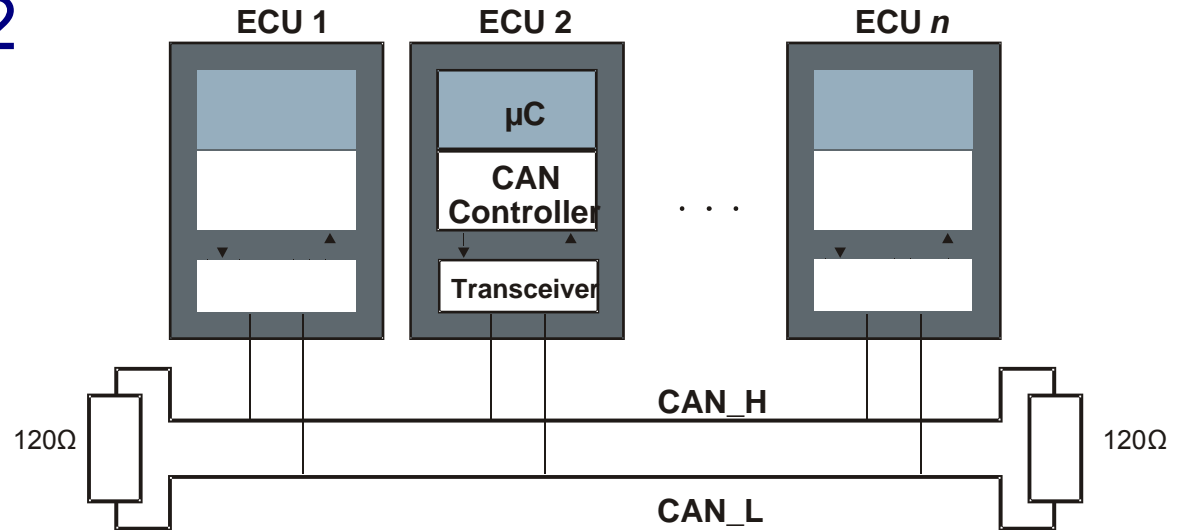
Controller Area Network (2)

- 1986, Bosch
- bus topology
- 2 bus levels:
 - low (dominant)
 - high (recessive)
- up to 110 subscribers
- max. 500 m at 125 kBit/s



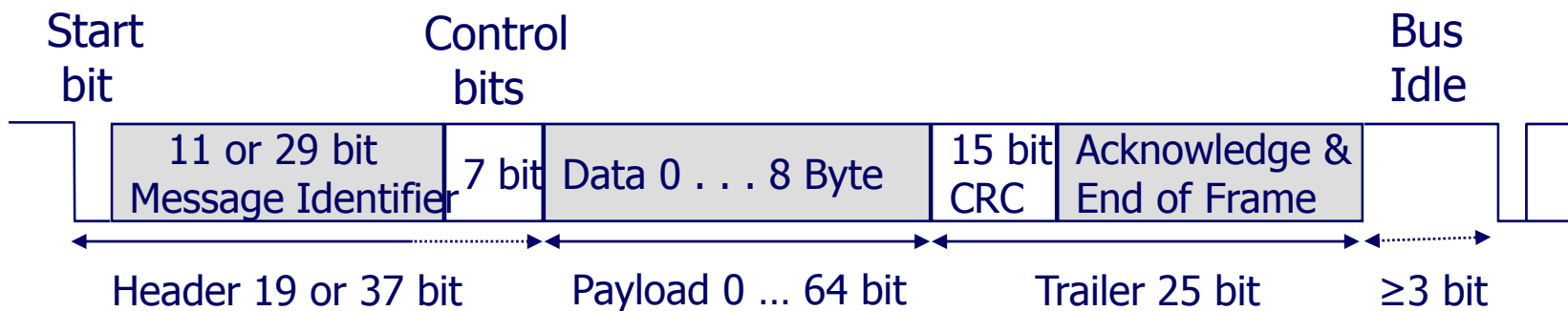
Controller Area Network (3)

- ISO 11898
 - Low-Speed-CAN (up to 125 kBit/s)
 - High-Speed-CAN (up to 1Mbit/s)
- OSI layers 1 and 2
- CANopen for upper layers



Controller Area Network (4)

- message oriented addressing
 - message ID with 11 (or 29) bits
- 4 frame formats: **data**, remote, error, overload
- data frame:



Medium access in CAN (1)

- random access, collision-free
- CSMA/CA with bus arbitration
- message oriented
 - message ID implies the message priority
- collision avoidance by priority-controlled bus assignment
- no destination addresses
 - broadcast/multicast

Medium access in CAN (2)

- by coding
 - “0” is dominant (low impedance)
 - „1“ is recessive (high impedance)
- bit stuffing: by sending 5 same bits 1 inverted “stuff” bit is inserted
- recessive bit on the bus if no station sends
- by receiving 3 recessive bits the bus is free
- bit-level synchronization by detecting the start bit (SOF)

Medium access in CAN (3)

- arbitration if a station wants to send
 - wait for 6 (recessive) bit times if a frame is actually transmitted
 - send the ID
 - sense the bus during the bit sending
 - if the actual (by another station) sended ID bit is unequal “my” correspondent ID bit then cancel the transmission and wait
 - the message with the highest priority (the most small ID) wins
- **non-preemptive** priority scheme
- **guaranteed** real-time transmission for the message with the highest priority!

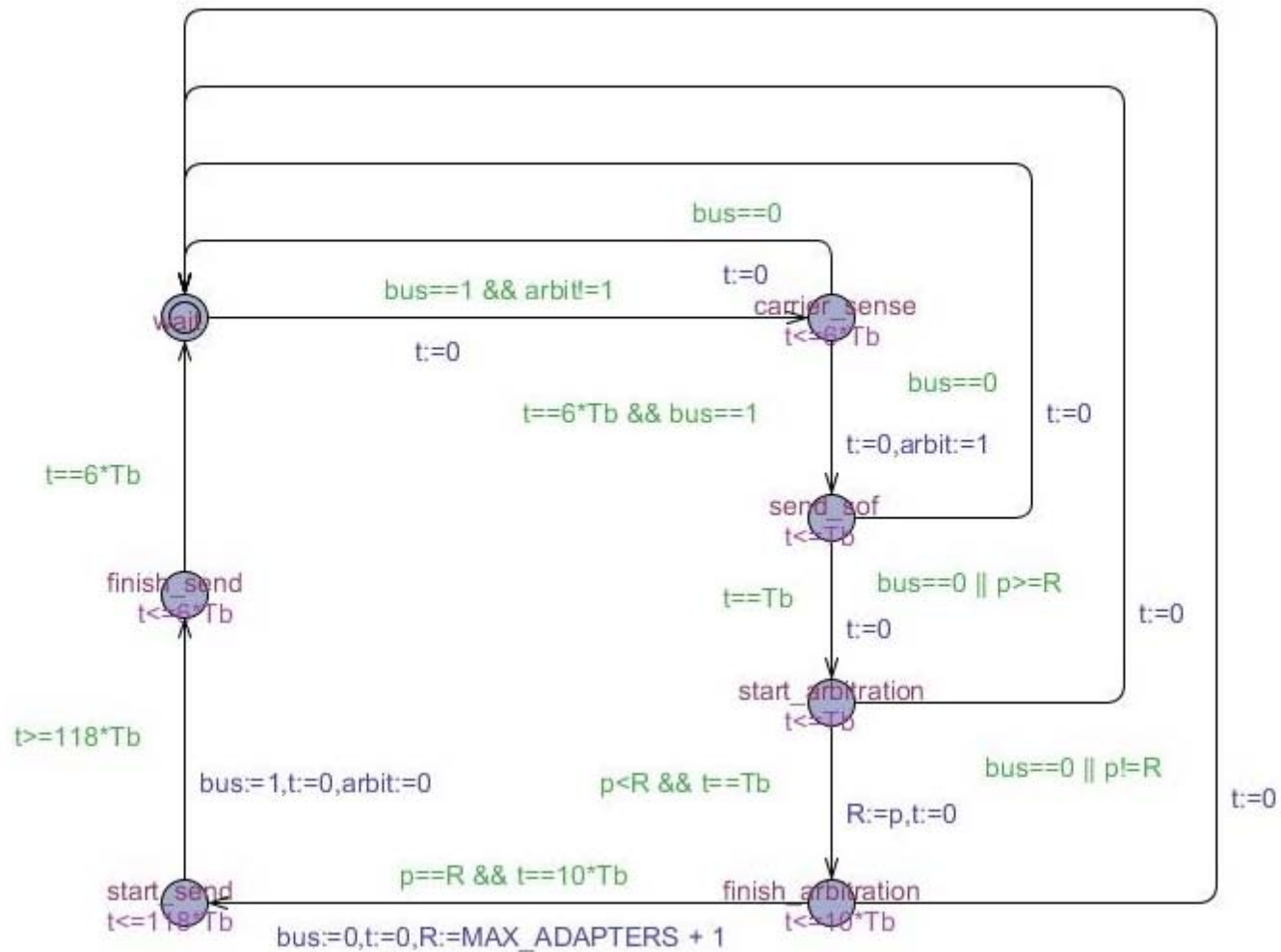
Modelling time

- modelling and verification of real-time systems
- **timed automata**: extend finite automata by a set of *clocks*
- transitions: *guards* (clock constraints) + *actions* + *reset* of clocks
- invariants: predicates over local clocks

Our CAN model (1)

- project: “*Verification of Real-Time Warranties in CAN*”
(partner: University of Erlangen-Nuremberg, Germany)
- goals (amongst others):
 - analysis of the CAN protocol
 - estimating delay bounds
- specification of CAN medium access by a timed automaton
- implementation in UPPAAL

Our CAN model (2)



Checking CAN properties (1)

- specification of CAN properties as **CTL** (Computation Tree Logic) formulas
- properties:
 - **safety** (something “bad” will never happen!)
 - **liveness** (something “good” will happen!)

Checking CAN properties (2)

- 1. property (safety):

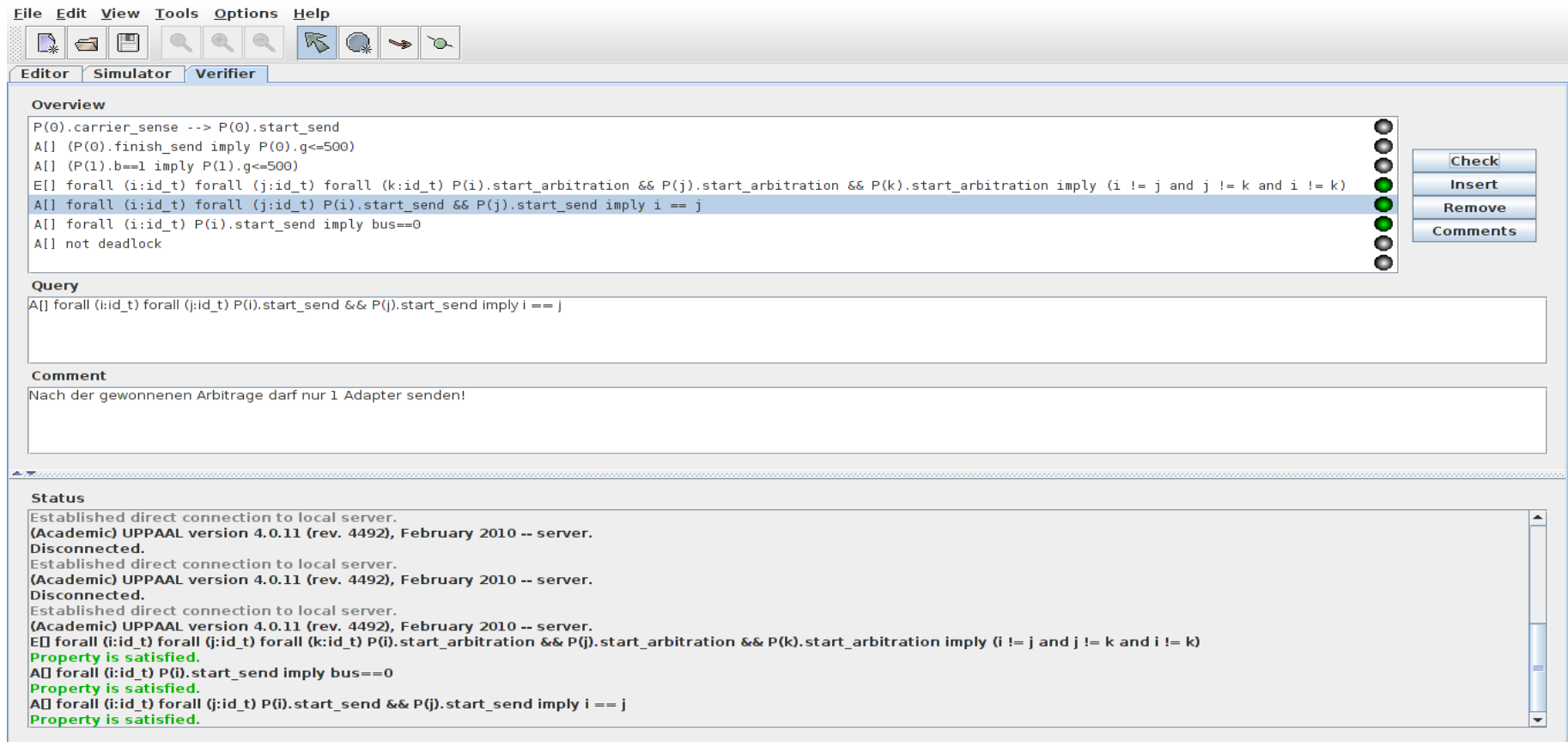
„at every time only 1 adapter may send (after winning the arbitration)”

- CTL formula (in UPPAAL):

$A[] \text{forall } (i:id_t) \text{forall } (j:id_t) P(i).start_send \ \&\& \ P(j).start_send$
 $\text{imply } i == j$

Checking CAN properties (3)

- proving this CTL formula (in UPPAAL):



The screenshot shows the UPPAAL Verifier interface with the following content:

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

```
P(0).carrier_sense --> P(0).start_send
A[] (P(0).finish_send imply P(0).g<=500)
A[] (P(1).b==1 imply P(1).g<=500)
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
A[] forall (i:id_t) forall (j:id_t) P(i).start_send && P(j).start_send imply i == j
A[] forall (i:id_t) P(i).start_send imply bus==0
A[] not deadlock
```

Query

```
A[] forall (i:id_t) forall (j:id_t) P(i).start_send && P(j).start_send imply i == j
```

Comment

Nach der gewonnenen Arbitrage darf nur 1 Adapter senden!

Status

```
Established direct connection to local server.
(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server.
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
Property is satisfied.
A[] forall (i:id_t) P(i).start_send imply bus==0
Property is satisfied.
A[] forall (i:id_t) forall (j:id_t) P(i).start_send && P(j).start_send imply i == j
Property is satisfied.
```

Checking CAN properties (4)

- 2. property (liveness):

“it can happen that all adapters try to use the bus concurrently”

- CTL formula:

$E[] \text{forall } (i:id_t) \text{forall } (j:id_t) \text{forall } (k:id_t) P(i).start_arbitration \ \&\& \\ P(j).start_arbitration \ \&\& P(k).start_arbitration \ \text{imply } (i \neq j \ \text{and } j \neq k \\ \text{and } i \neq k)$

Checking CAN properties (5)

- proving this CTL formula:

The screenshot shows the UPPAAL Verifier interface with the following content:

Overview

```
P(0).carrier_sense --> P(0).start_send
A[] (P(0).finish_send imply P(0).g<=500)
A[] (P(1).b==1 imply P(1).g<=500)
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
A[] forall (i:id_t) forall (j:id_t) P(i).start_send && P(j).start_send imply i == j
A[] forall (i:id_t) P(i).start_send imply bus==0
A[] not deadlock
```

Query

```
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
```

Comment

In der Arbitrierung können sich mehrere (z.B. 3) Adapter befinden!

Status

```
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server.
Disconnected.
Established direct connection to local server.
(Academic) UPPAAL version 4.0.11 (rev. 4492), February 2010 -- server.
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
Property is satisfied.
A[] forall (i:id_t) P(i).start_send imply bus==0
Property is satisfied.
A[] forall (i:id_t) forall (j:id_t) P(i).start_send && P(j).start_send imply i == j
Property is satisfied.
E[] forall (i:id_t) forall (j:id_t) forall (k:id_t) P(i).start_arbitration && P(j).start_arbitration && P(k).start_arbitration imply (i != j and j != k and i != k)
Property is satisfied.
```

Checking CAN properties (6)

- other properties:

- deadlock freeness:

$A[]$ not deadlock

- „*the adapter with the highest priority will always make a transition from the state `carrier_sense` to the state `start_send`*“:

$P(0).carrier_sense \rightarrow P(0).start_send$

- no **livelock** checking in UPPAAL!

Delay bounds (1)

- quantitative properties, real-time liveness
- in UPPAAL: additional clock variable z + Boolean variable b :

$$A[](b \implies z \leq t)$$

- example:

„is it guaranteed that the adapter with the highest priority will always finish with frame sending within 136 bit times?“

Delay bounds (2)

- CTL formula:

$A[] (P(0).b==1 \text{ imply } P(0).g \leq 272)$

- proving this CTL formula:

The screenshot shows the UPPAAL model checker interface. The main window is divided into several sections:

- Overview:** Contains the model description and the CTL formula being verified: $A[] (P(0).b==1 \text{ imply } P(0).g \leq 272)$. It also lists other properties: $E[] \text{ forall } (i:id_t) \text{ forall } (j:id_t) \text{ forall } (k:id_t) P(i).start_arbitration \ \&\& \ P(j).start_arbitration \ \&\& \ P(k).start_arbitration \ \text{imply } (i \neq j \ \&\& \ j \neq k \ \&\& \ i \neq k)$, $A[] \text{ forall } (i:id_t) \text{ forall } (j:id_t) P(i).start_send \ \&\& \ P(j).start_send \ \text{imply } i == j$, $A[] \text{ forall } (i:id_t) P(i).start_send \ \text{imply } bus==0$, and $A[] \text{ not deadlock}$.
- Query:** Shows the CTL formula being verified: $A[] (P(0).b==1 \text{ imply } P(0).g \leq 272)$.
- Comment:** Contains the text: "Adapter X beendet das Senden des gesamten Rahmens in weniger als Y Zeiteinheiten."
- Status:** Shows the verification results: "Property is satisfied." for all properties, and "Disconnected." for the local server connection.

Delay bounds (3)

- the same property for all other priorities will **not** be fulfilled! (no guarantees for delay bounds of other adapters)
- if we introduce **cycles** by sending frames then such delay bounds can be estimated for **all** adapters!

Summary

- only one timed automaton for CSMA/CA with bus arbitration (very compact model!)
- extending our automaton by cycles **all delay bounds** could be estimated
- proving **qualitative properties** of the model
- but: **low efficiency** of formal verification! (only 4 adapters)