Modelling the network
Embedded Systems Specification and Design

David Kendall

Northumbria University
How to model a Controller Area Network in UPPAAL

Goal:
- A general, flexible model of CAN that can be incorporated with a variety of process models for simulation and verification of properties

Example based on [DBB07] to show its utility
Preliminaries

CAN Frame

Example system

<table>
<thead>
<tr>
<th>Message</th>
<th>Priority</th>
<th>Period</th>
<th>Deadline</th>
<th>TX time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2.5 ms</td>
<td>2.5 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>3.5 ms</td>
<td>3.25 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3.5 ms</td>
<td>3.25 ms</td>
<td>1 ms</td>
</tr>
</tbody>
</table>
\texttt{const int dlb[3] = \{4, 4, 4\};}
\texttt{const int dub[3] = \{4, 4, 4\};}
\texttt{const int dB[3] = \{0, 0, 0\};}
\texttt{const int duB[3] = \{0, 0, 0\};}
\texttt{message_t m = MIN_INVALID_CAN_ID;}
\texttt{queue_t q = \{MIN_INVALID_CAN_ID, MIN_INVALID_CAN_ID, MIN_INVALID_CAN_ID\};}
\texttt{clock c;}

David Kendall (Northumbria University)  Modelling the network  4 / 14
A computing node periodically enqueues its message.

The notional *release* of the message is represented by the synchronisation action $r!$.

The jitter in the software task that is responsible for enqueuing the message is modelled by the interval $[0, JITTER]$. 

```plaintext
Node(const message_t m, const int PERIOD, const int JITTER)

clock p;
clock j;
```
The observer examines every message that is enqueued

If an enqueued message is the one of interest the observer waits for the message to become available for acceptance

If this happens before the deadline, all is well; otherwise ERROR
This is an auxiliary process that is always prepared to engage in a urgent action $u$.

$u$ – ensures that enqueued messages begin transmission without delay when the channel is free.
Global declarations

```c
const int MIN_INVALID_CAN_ID = 3;
const int QSIZE = MIN_INVALID_CAN_ID;

typedef int queue_t[QSIZE];
typedef int [0, MIN_INVALID_CAN_ID] message_t;
typedef int [0, QSIZE - 1] qindex_t;

broadcast chan r;
chan e;
broadcast chan a;
urgent chan u;

message_t tmp = MIN_INVALID_CAN_ID;

bool is_empty(queue_t q) {
    return forall (i : qindex_t) q[i] == MIN_INVALID_CAN_ID;
}
```
void enq(queue_t& q, message_t m) {
    q[m] = m;
}

message_t deq(queue_t& q) {
    message_t result = MIN_INVALID_CAN_ID;
    for (i : qindex_t) {
        if (q[i] != MIN_INVALID_CAN_ID) {
            result = q[i];
            q[i] = MIN_INVALID_CAN_ID;
            return result;
        }
    }
    return MIN_INVALID_CAN_ID;
}
System declarations

// System declarations

K = CanChannel();

// Node(m, p, j) releases message m periodically with period p
// and queueing jitter j
A = Node(0, 10, 0);
B = Node(1, 14, 0);
C = Node(2, 14, 0);

// Simple observers
// Only reliable if DEADLINE <= PERIOD
O_A = RTobserver(0, 10);
O_B = RTobserver(1, 13);
O_C = RTobserver(2, 13);

// The system should have a CAN channel, one node per message,
// an observer for a message of interest and
// an Aux process for urgent actions.
system K, A, B, C, O_C, Aux;
The verifier can be used to check the message response time
It immediately reveals the error in Tindell’s original analysis . . .
. . . and easily allows the discovery of the correct response time
The property of interest is $E<> O_C.ERROR$
Note $E<> O_C.ERROR$ is *true if and only if* $A[]$ not $O_C.ERROR$ is *false*
Reasoning about jitter in a distributed system

- Jitter in some system response is the difference between the longest time and the shortest time between the occurrence of the event marking the start of the response and the event marking the end of the response.
- Taking account of jitter when reasoning about the behaviour of a distributed embedded system is a challenging problem.
- This model shows one of the simpler approaches.
// System declarations

K = CanChannel();

// Node(m, p, j) releases message m periodically with period p
// and queueing jitter j
A = Node(0, 10, 2);
B = Node(1, 14, 0);
C = Node(2, 14, 0);

// Simple observers
// Only reliable if DEADLINE <= PERIOD
O_A = RTObserver(0, 10);
O_B = RTObserver(1, 13);
O_C = RTObserver(2, 14);

// The system should have a CAN channel, one node per message,
// an observer for a message of interest and
// an Aux process for urgent actions.
system K, A, B, C, O_B, Aux;
Discovering the effects of jitter

- This model shows that jitter in the release time of message A can have a detrimental effect on the response time of messages B and C.
- Check the property $E<> O_B \cdot \text{ERROR}$ with a variety of deadlines for $O_B$ to discover the new worst-case response time for B messages.