A Formal Model of an AUTOSAR In-Vehicle Communications Protocol

San Choosang, Rattapat Taburan and Steven Gordon
Sirindhorn International Institute of Technology
Thammasat University
Bangkadi, Thailand 12000
Email: san_choosang@hotmail.com, jaecibake@hotmail.com, steve@siit.tu.ac.th

***Abstract*—AUTOSAR is an architecture for developing component-based software applications for vehicles. It aims to allow control, diagnostic and safety applications to be developed independent of the vehicle Electronic Control Units, as well as the vehicle communication bus technology (e.g. FlexRay, CAN). This paper presents a formal Coloured Petri net (CPN) model of the AUTOSAR FlexRay Transport Protocol. After defining the expected terminal states of the protocol, state space analysis of the model is used to prove the protocol is free of unexpected deadlocks. In addition it is shown that the protocol faithfully refines the service offered to the higher layer, and that transmit/receive buffers have an upper bound of frames dependent on the block size and retry limits. The CPN model not only serves as a mean of verifying functional properties of the FlexRay Transport Protocol, but can be adapted to evaluate the protocol performance for vehicle applications.**

***Index Terms*—formal verification, communication protocols, automotive applications, Coloured Petri nets**

I. Introduction

Vehicles today contain 10's to 100's of embedded computers, or Electronic Control Units (ECUs), that together control the engine, airbags, doors, seats, as well as provide information to other on-board devices (e.g. telephone, entertainment systems) and to users (e.g. via an On-Board Diagnostics interface). The increasing number and complexity of applications using ECUs has contributed to the development of new in-vehicle communications systems, as well as techniques to simplify the development of applications utilising multiple ECUs. For the former, FlexRay [1] is a potential replacement for Controller Area Network (CAN), a bus for inter-ECU communications in many vehicles today. For the latter, automobile manufacturers are developing the Automotive Open Systems Architecture (AUTOSAR) [2] to allow software components to communicate irrespective of the ECU or bus technology (FlexRay, CAN) that is in use. A specific part of AUTOSAR is the FlexRay Transport Protocol [3], which allows software components to send longer messages with higher reliability than when directly using the FlexRay bus.

This paper presents a formal model of the FlexRay Transport Protocol, as well as analysis results. Formal modelling and analysis of communication protocols, or protocol verification [4], [5], is important in system design stages as functional errors discovered during testing and usage are expensive to fix. More importantly in vehicles, unexpected behaviour in communications systems may lead to fatalities. The aim of our research is to verify the functional correctness of the FlexRay Transport Protocol. The key contributions of this paper are the development of a Coloured Petri net (CPN) [6] model of the protocol, as well proof, using state space analysis, of absence of deadlocks in selected configurations of the protocol. The proof is a first steps towards full verification of the FlexRay Transport Protocol.

The contents of this paper are: Section II provides an overview of AUTOSAR and the FlexRay Transport Protocol. Section III introduces CPNs and explains how they are used in protocol verification. Section IV presents our CPN model of the protocol. State space analysis results proving the absence of deadlocks are given in Section V, followed by concluding remarks in Section VI.

II. In-Vehicle Communications Systems

A. AUTOSAR and FlexRay

A vehicle typically consists of 10's of ECUs, as well as sensors and actuators, connected via one or more communication buses. AUTOSAR [2] is an architecture that aims to simplify the development of component-based in-vehicle software applications by hiding the complexities of the ECU's and communications bus from components. Software components communicate via a *Virtual Functional Bus* (VFB) as shown in Fig. 1.

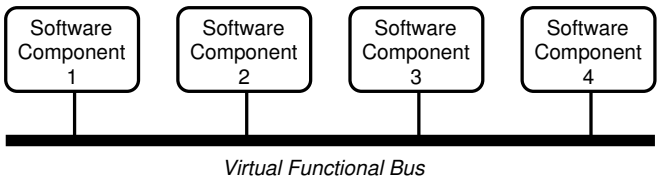

Fig. 1. AUTOSAR communication via a Virtual Functional Bus

The VFB is implemented by one *Run Time Environment* (RTE) per ECU. AUTOSAR specifies standard interfaces between the RTE and the ECU operating system, device drivers, communication mechanisms and microcontroller. In AUTOSAR these are referred to as the Basic Software as shown in Fig. 2, and in further detail in Fig. 3.

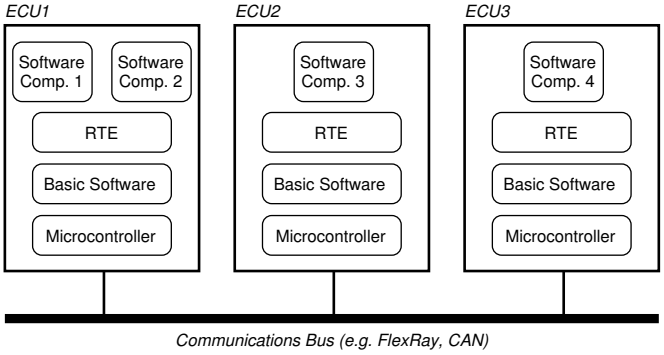

Fig. 2. Structure of ECU's in AUTOSAR

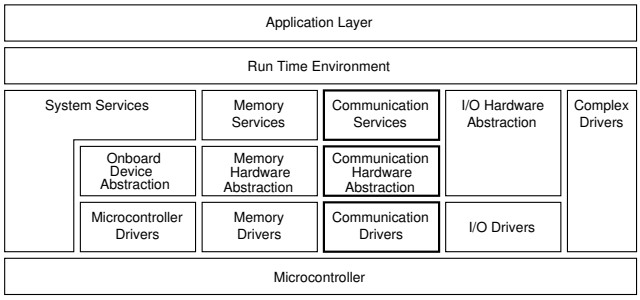

Fig. 3. AUTOSAR Layered Software Architecture

Focussing on the communication mechanisms (highlighted in Fig. 3, with selected blocks shown in Fig. 4), data to be sent between software components is delivered by the AUTOSAR Communications (COM) layer to the *PDU Router*. AUTOSAR COM provides an communications API for applications—it is based on OSEK/VDX COM [7]. The PDU Router determines the technique to transport data between software components (e.g. selects from different bus standards such as FlexRay and CAN, as well as features required).

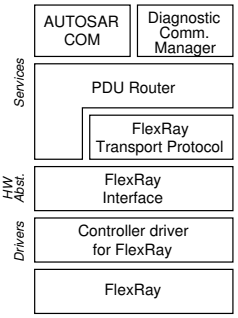

Fig. 4. AUTOSAR FlexRay Layered Architecture

If FlexRay is selected by the PDU Router, then the FlexRay Transport Protocol may be used to provide reliability and efficiency features not offered by the FlexRay bus (discussed further in Section II-B). Below the communication services is the FlexRay interface which provide an abstract interface to the FlexRay drivers and FlexRay controller (which may be implemented internal or external to the microcontroller).

The focus of this research is on the FlexRay Transport Protocol within the AUTOSAR architecture. Many of the blocks in AUTOSAR are based on existing standards/protocols/software and have been used/tested extensively. The FlexRay Transport Protocol is in fact based on an existing CAN/ISO transport protocol. However several significant extensions are introduced which warrants in-depth, formal analysis of its operation.

B. FlexRay Transport Protocol in AUTOSAR

FlexRay [1] defines a physical and data link layer protocol for distributed bus-based communication between a set of controllers. The physical layer allows each controller to connect via one or two channels at data rates of 10Mb/s. The data link layer uses TDMA allowing each controller to transmit 255 bytes of data in a frame per time slot.

The FlexRay Transport Protocol (FrTp) provides both a confirmed and unconfirmed communications service for AUTOSAR applications that use a FlexRay bus. It is partly based on ISO 15765-2 [8], a standard for unconfirmed communications for diagnostic applications in a CAN-based vehicle. In addition FrTp provides extra reliability features (e.g. acknowledgements and retries) to offer confirmed communications. Segmentation and flow control are also implemented to improve performance. The error/flow control scheme is based on Go-Back-N. This research, and subsequent description, focusses on the confirmed service, as it is a substantially different from and more complex than the existing ISO 15765-2

protocol. Only features relevant to the modelling/analysis tasks are described; for a full treatment of FrTp see [3].

1) FrTp Service Specification: Application and/or diagnostics data is sent via AUTOSAR COM or DCM to the PDU Router. If required, the PDU Router delivers that data to FrTp, which then transfers the data to the destination ECU, PDU Router and eventually application. The service provided by FrTp to the user (PDU Router) is informally described in Section 5 of [3]. A set of service primitives are defined, as shown in Fig. 5.

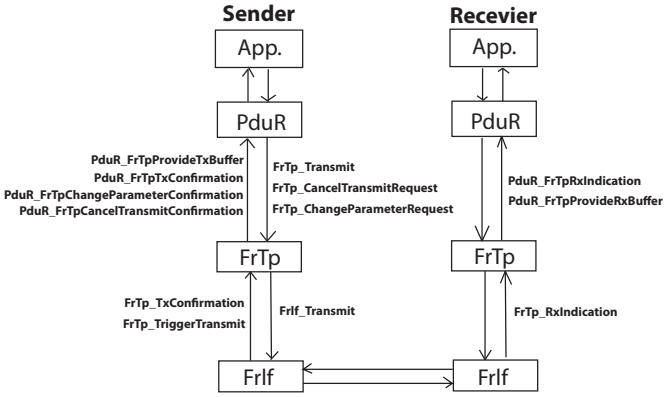

Fig. 5. FrTp Service Primitives

The key primitives for data transfer are:

1) FrTp_Transmit is called by PDU Router to initiate data transmission at the Sender.
2) The FlexRay Interface (FrIf) is used to transmit that data to the Receiver FrTp.
3) Upon successfully receiving the data, the Receiver passes that data to the PDU Router via PduR_FrTpRxIndication. In addition a confirmation is sent to the Sender and passed to the PDU Router as PduR_FrTpTxConfirmation(successful).
4) Unsuccessful delivery of data results in no indication to the Receiver PDU Router, but a PduR_FrTpTxConfirmation(unsuccessful) is delivered to Sender PDU Router.

Other primitives are used for optional features such as cancelling a transmission. Our analyse does not yet consider these optional features.

Although [3] defines the set of service primitives, there is no formal definition of the valid sequences of primitives. We have defined a set of sequences based on our understanding of the protocol and AUTOSAR architecture. Fig. 6 shows the set of sequences as a FSA.

2) FrTp Protocol Specification: The FrTp protocol specification is given in sections 7, 8 and 9 of [3]. Data can be sent using different methods with FrTp:

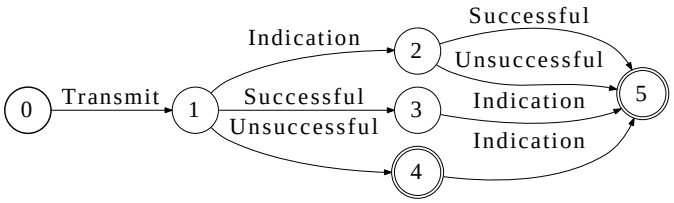

Fig. 6. Desired FrTp Service Language

1) Unacknowledged segmented or unsegmented transfer
2) Acknowledged segmented or unsegmented transfer, but no retries
3) Acknowledged segmented or unsegmented transfer with retries
4) Unacknowledged segmented or unsegmented multicast transfer

The first method is identical to ISO 15765-2, while the other three are new. This paper addresses only the unicast acknowledged transfer (methods 2 and 3).

Different frame types are used in FrTp. A Single Frame (SF) is used to carry data when the data to send does not exceed the payload capacity of a FlexRay frame. A First Frame (FF) is the first frame sent when segmentation is needed (data to send is greater than payload capacity). Consecutive Frames (CF) are sent after the FF. In the return path Flow Control (FC) frames are sent by the receiver to control the sending rate: CTS value is set if the receiver is ready to receive; WAIT value is set if the sender must wait; and OVERFLOW value is set if receiver has exhausted its buffer. Finally, a Positive Ack (PACK) indicates all data has been successfully received, while a Negative Ack (NACK) indicates an error (and the transfer should be aborted).

An example of FrTp with segmented, acknowledged transfer with retries is illustrated in Fig. 7. The FF is transmitted and upon receipt the Receiver replies with a FC frame indicating the Sender is clear to send the next frames. Included in the FC frame is the Block Size (BS) which indicates the number of CFs the Sender may send before waiting for the next Ack/CTS.

If a frame is lost or arrives with errors, the receiver sends a NACK indicating the frame that is missing (all frames contain sequence numbers). Upon receiving the NACK the sender must retransmit all frames sent but not yet acknowledged. This is a Go-Back-N error/flow control scheme. Once all data is received by the Receiver (the total data length is included in the First Frame), a PACK is sent indicating the successful completion of data transfer.

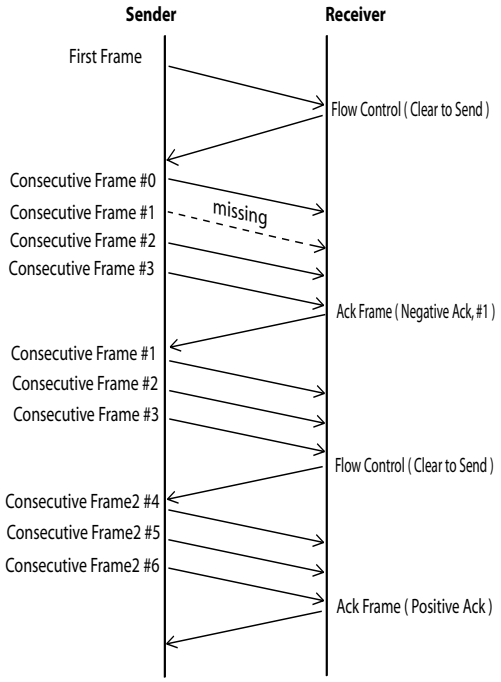

Fig. 7. FrTp Message Sequence Diagram

III. Coloured Petri Nets for Protocol Verification

Petri nets are a formal method that are well suited to the analysis of distributed systems, including protocols, because of their ability to express concurrency, non-determinism and system concepts at different levels of abstraction. They have an underlying mathematical definition, therefore allowing for proof of static and dynamic properties of the system modelled, as well as a graphical notation that, with computer tool support, allows non-experts to apply them to the modelling of practical systems. CPNs [6] are a class of high-level nets that extend the features of basic Petri nets. Section III-A provides an informal definition of CPNs, while Section III-B describes their role in verifying protocols, as well as the related work in this area.

A. Coloured Petri Nets

CPNs are directed graphs with two types of nodes: a set of *places*, P, and a set of *transitions*, T. These nodes are normally illustrated as ellipses and rectangles, respectively. Directed arcs can only be from place to transition (*input arcs*) or transition to place (*output arcs*). Directed arcs can only be between nodes of different types. Places are typed by a *colour set*. The colour set determines the type of values that can mark a place. These values are called *tokens*. A multiset of tokens in a place p is called its marking ($M(p)$) and the marking of the CPN comprises the marking of all places (M). Transitions and arcs can also have inscriptions (expressions).

The execution of a CPN consists of occurrence of transitions. A transition can occur if it is *enabled*, and it is enabled if: for all input places, sufficient tokens exist that satisfy the input arc inscriptions; and the transition inscription (or *guard*) evaluates to true.

Variables, which are local to a transition, may be used in inscriptions. The values they are bound to on occurrence of a transition give, together with the transition name, a *binding element*. When a transition occurs, tokens required by the input arcs are removed from the input places, and the evaluation of the expression on the output arcs give the tokens to be added to the output places.

As an example consider the transition FrTp_Indication in Fig. 10. There are input arcs from places Data_Received and Indication_Sent. Considering the inscriptions on these arcs, as well as the transition guard (enclosed in [and]), this transition is enabled if:

1) $M(\text{Data\_Received}) = (x, x, \_)$ where $x <> 0$
2) and $M(\text{Indication\_Sent}) = \text{true}$

The occurrence of this transition will remove tokens from the input places and create 1 Indication token in To_Rx_PduR and 1 true token in Indication_Sent.

The main benefit of creating formal models is that certain properties of the models can be proved. State space analysis is one method for doing this formal analysis. A state space of a CPN is a directed graph with nodes representing the marking (or state) of the CPN and arcs representing the binding elements (or state changes or events). By generating the complete state space from some initial marking properties such as reachability, deadlocks, livelocks and bounds can automatically be proven for the system modelled. The state explosion problem is however a significant limitation to this formal analysis approach when analysing complex protocols.

B. Protocol Verification Methodology

As stated early, it is important to verify that communication protocols used within vehicles operate correctly. To do so, two sets of properties of the protocol must be considered.

Firstly, there are dynamic properties of a protocol that are expected from its correct operation. Common desired properties include the absence of deadlocks and livelocks in the protocol. Other properties may be specific to the protocol, such as the number of messages stored in a

receive buffer never goes above a pre-defined limit. State space analysis is well-suited to automatically proving a desired set of properties of a protocol.

The second set of properties of interest are the interactions between the protocol and the user, where the user may be another protocol, a software component or (less likely) a human. The users requirements of the protocol, or *service specification*, should define the set of possible interactions between user and protocol. The interactions are referred to as *service primitives* and the set of possible orderings the *service language*. Protocol verification aims to prove that the protocol faithfully implements the desired service specification. If the desired service language is known (in some cases it is specified in the protocol standard, in other cases it can be derived from related information), the state space of the protocol can be used to verify the faithful refinement of service. The protocol state space defines the set of interactions between protocol and user, as well as detailed protocol operations (e.g. timeouts, message transmission and reception). Treating the state space as a deterministic finite state automata, and then hiding the detailed protocol operations (i.e. considering all events that are not interactions with the user as ϵ-transitions in the FSA), the minimised deterministic FSA can be obtained (i.e. the *protocol language*) and easily compared to the desired service language. If the protocol and service languages are identical, then it is verified that the protocol faithfully refines the desired service.

The steps for proving the properties described above are part of a commonly applied protocol engineering methodology [5]. Although different methods can be used, Coloured Petri nets are well suited to the task, as has been demonstrated in numerous examples [9], [10], [11], [12]. The steps applied in this research are summarised in Fig. 8.

Numerous researchers have considered AUTOSAR design issues, including automated testing [13], [14] and hardware-in-the-loop analysis [15], but to date there has been no formal analysis of the FlexRay Transport Protocol. Two works similar to our research are [16] and [11]. [16] uses CPNs to model and evaluate the performance of a CAN bus. A timed CPN is used to measure throughput and latency in a simple CAN network. However [16] is not attempting formal verification of the protocol, nor focussing on AUTOSAR or FlexRay. [11] presents extensive formal verification of a class of stop-and-wait protocols. Properties of the protocols are proved for arbitrary parameter values (unlike this paper which addresses analyses only several values).

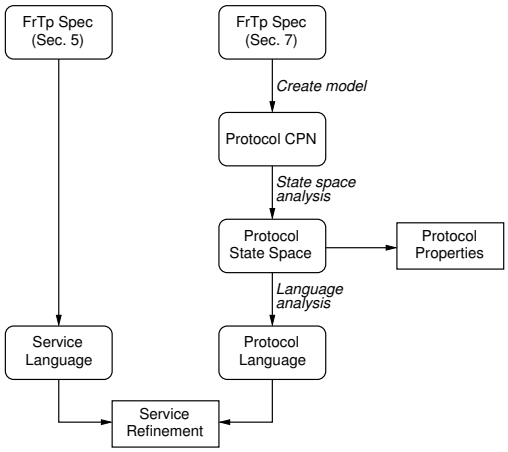

Fig. 8. Protocol Verification Methodology for FrTp

However our work considers a different, more complex protocol (Go-Back-N based, instead of stop-and-wait) and includes detailed modelling of the FrTp interaction with the higher layer.

C. Tool Support

CPN Tools[1] is the most popular tool for creating and analysing CPNs. It supports graphical editing of the model, simulation to validate the design, and state space analysis. Other features include support for timed CPNs, external visualisations and advanced state space reduction techniques. For inscriptions CPN Tools uses a variant of the functional programming language Standard ML called CPN ML. CPN ML is also used to perform queries on the state space to automatically prove properties of the model.

Using CPN ML queries, the state space can be saved in a textual format suitable for analysis using AT&Ts FSM Library[2]. This is used to generate the protocol language and comparison with the service language.

IV. CPN Model of FlexRay Transport Protocol

In creating the FrTp CPN model the following assumptions are made:

1) The optional features of cancelling a transmission and changing parameters are not modelled.
2) The FlexRay communications bus guarantees ordered delivery (i.e. FIFO queue). Both cases of a reliable and unreliable (lossy) bus are modelled.
3) Basic ISO-compliant frame types are used (as opposed to optional extended frames).

[1]http://wiki.daimi.au.dk/cpntools/

[2]http://www.research.att.com/~fsmtools/fsm/

```
(* Default values of system constants *)
val WholeDataC = ref 100;
val FrameSizeC = ref 100;
val BSC = ref 1;
val InitialSNC = ref 0;
val MaxRetryC = ref 1;
val TimeoutEnableC = ref true;
val LossEnableC = ref true;
val OVFLWenableC = ref true;
(* Data types (colour sets *)
colset FrameType = with SingleFrame | FirstFrame |
       ConsecutiveFrame | FlowControl | AckFrame;
colset FrameState = with CTS | WT | OVFLW |
       FrameStateNotSet;
colset Ack = with Positive | Negative | AckNotSet;
colset Frame = record ft:FrameType *
                      dl:DataLength *
                      fs:FrameState *
                      sn:SN *
                      ack:Ack *
                      bs:BlockSize *
                      data:Data;
colset Frames = list Frame;
colset DataRemain = product BytesLeft * SentTmp * SN;
colset DataInfo = product INT*INT*INT*INT*INT;
colset NSDU = with Transmit | Indication | Successful |
       Unsuccessful;
colset DataReceived = product DataLength * RxdFrame * SN;
colset BSxSN = product INT*INT;
colset Timeout = product INT*INT*INT*INT;
(* colsets not listed are of type int *)
(* variable and function definitions are not shown *)
```

Fig. 9. Selected FrTp CPN Declarations

A. FrTp Protocol CPN

FrTp is modelled as a hierarchical CPN, with the top-level page shown in Fig. 10, and sub-pages in Figures 11, 12 and 13. Important constants and data types (colour sets) used in the model are in Fig. 9.

The flow of data for a single message sent from Sender to Receiver is highlighted by the thick arcs in the top-level page. On the left of the page is the Sender, in the middle is the communications channel, and the right is the Receiver. At the top the interface between PDU Router and FrTp is modelled. The initial marking of From_Tx_PduR is 1‘Transmit. This indicates the PDU Router has data ready to transmit. The occurrence of FrTp_Transmit models the delivery of that data to FrTp.

The actual data is not modelled (it does not affect the operation of the protocol): instead the model records the size of the data (in bytes). The place Data_to_Send contains a 5-tuple: bytes remaining to be acknowledged; bytes remaining to be sent; sequence number of of next frame to be acknowledged; sequence number of next frame to be sent; and current block size. This information is used and updated by transitions modelling the generation of frames, as well as reception of frames (in Fig. 12).

Frames containing data are generated and stored in the Tx buffer (Sender_FRTP). This, as well as the Rx buffer (Receiver_FRTP), is modelled as a FIFO queue. The type of frame generated (Single Frame, First Frame, Consecutive Frame) depends on the data size, block size and previous frames generated. The inscription on the arc from Generate_Frame to Sender_FRTP contains CPN ML functions that generate the appropriate frame.

The communications channel is modelled as a full-duplex, ordered medium. In addition, both the case of reliable and unreliable (lossy) channel are modelled. Using the standard CPN ML constructs for modelling FIFO queues, a list of frames is stored in Data_Channel. Initially empty, the occurrence of TxFrame adds a frame to the tail of the list and RxFrame extracts the frame from the head of the list. The function LossEnable() returns true if a model input configuration variable is set to enable frame loss. In that case, transition Packet_Loss will be enabled whenever a frame is in Data_Channel, and its occurrence deletes a frame from the head of the list. As the focus is on functional properties of the protocol (not performance), loss is non-deterministic: any frame may either be lost or received.

Upon receiving a data frame, the Received processes the received frame. The detailed procedure is modelled on a sub-page (Fig. 11) represented by the transition Process_Rxd_Frame. The place Data_Received models the total number of bytes expected, the bytes received, and the next expected sequence number in a 3-tuple. Once the bytes received equals the bytes expected the data can be delivered to the Receiver PDU Router. This is modelled by the FrTp_Indication transition, putting an Indication token in To_Rx_PduR.

As data frames are processed by the Receiver (Fig. 11), FC, NACK and PACK frames may be sent on the return Ack channel. Upon reception the Sender processes the frame, as modelled on sub-page Rx_Ack. If a FC frame is received, depending on the number of retries (Number_of_Retry, new data frames may be generated and sent. If a PACK is received, then the data transfer is successful and via the PduR_FrTpTxConfirmation sub-page, a Confirmation(Successful) service primitive is delivered to the PDU Router. If a FC(Overflow) is received, or the maximum number of retries has been reached, a Confirmation(Unsuccessful) primitive is delivered to the PDU Router.

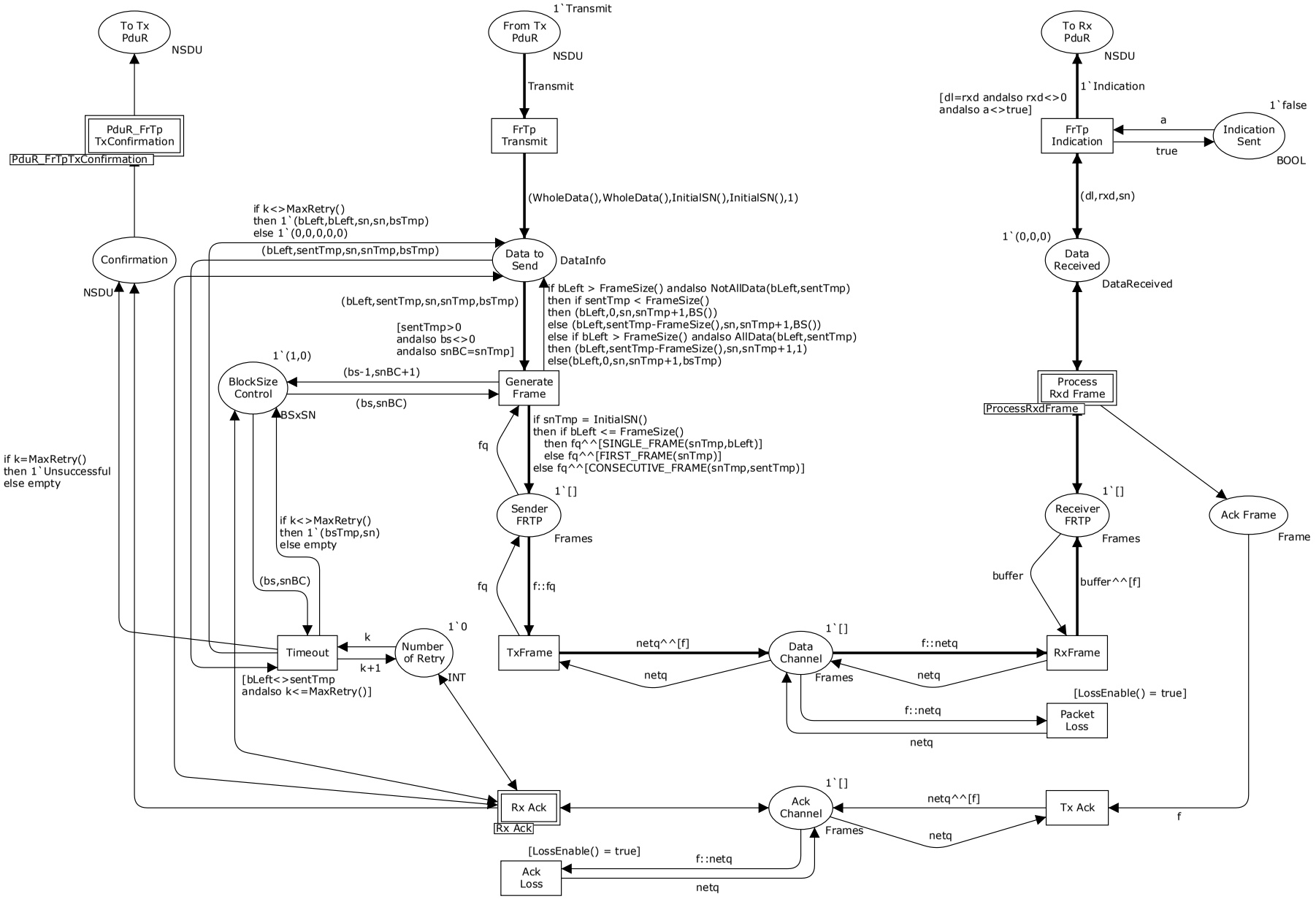

Fig. 10. FrTp CPN Model: Top-level page

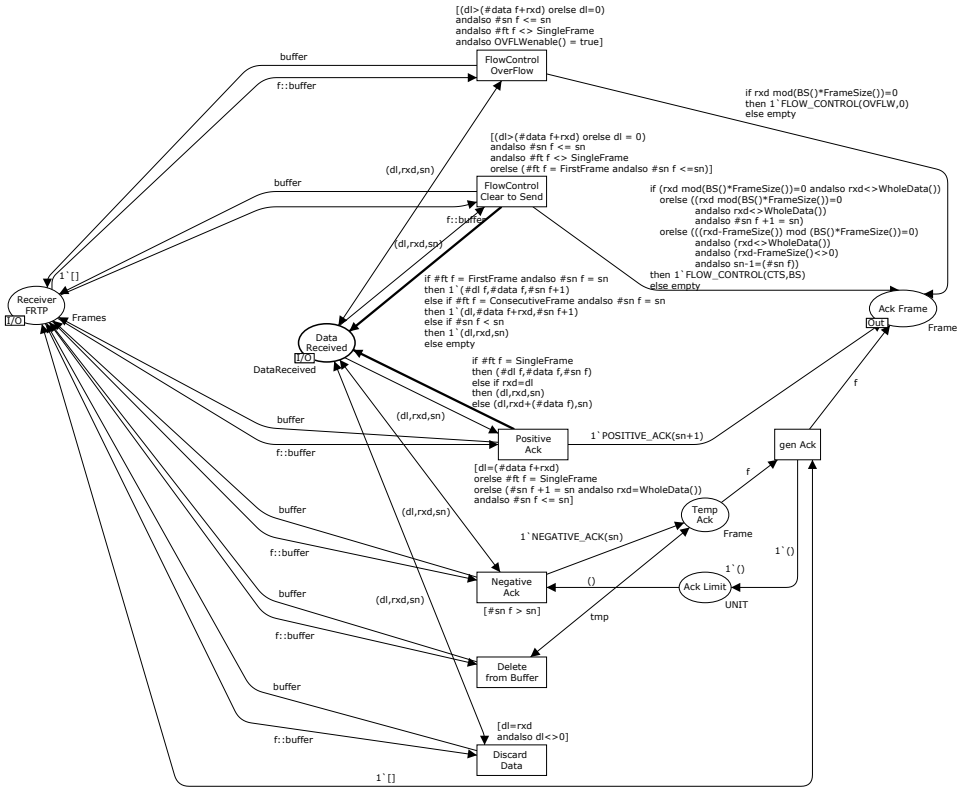

Fig. 11. FrTp CPN Model: ProcessRxdFrame

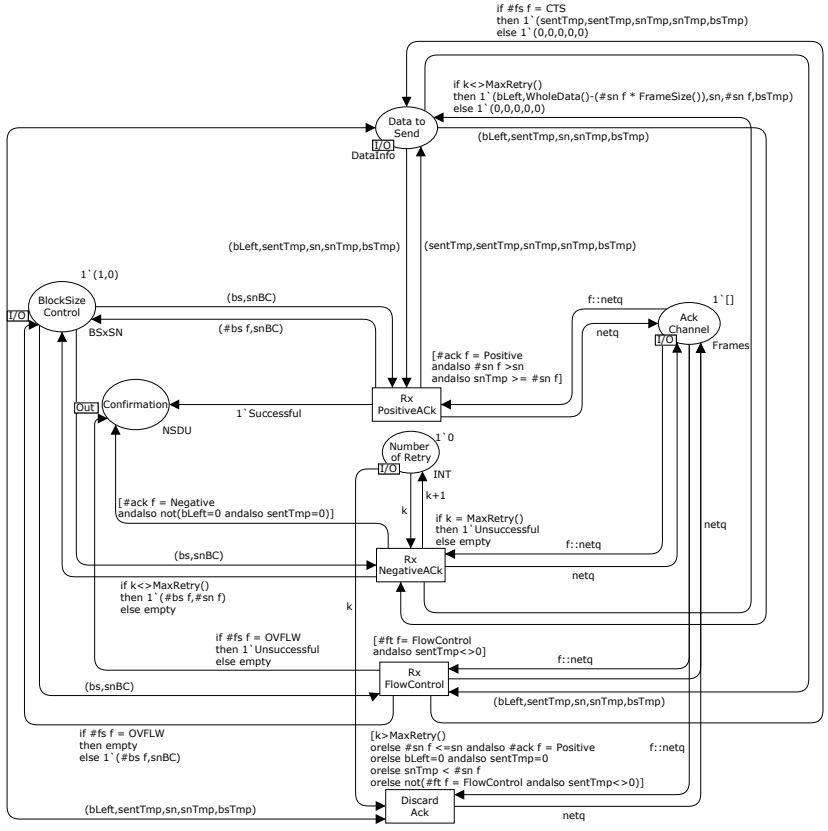

Fig. 12. FrTp CPN Model: RxAck

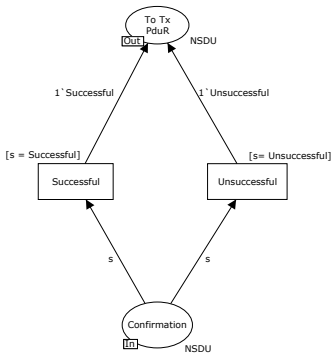

Fig. 13. FrTp CPN Model: TxConfirmation

Time is not explicitly modelled in the CPN. The occurrence of the Timeout transition is non-deterministic: if data has been sent, the timeout may or may not occur before the ACK is received.

B. FrTp Model Parameters

Key input parameters to the model are:

- WD (Whole Data) size of the data sent by the PDU Router [bytes]
- FS (Frame Size) maximum size of payload in a frame [bytes]
- BS (Block Size) number of blocks Receiver allows to be sent before FC
- MR (Max Retry) maximum number of retries Sender makes before aborting the transfer
- Loss if true then frame loss is possible in the channel; otherwise a reliable channel is assumed

Varying the values of these parameters allows for investigation of FrTp under different conditions.

V. Formal Analysis of FrTp

Analysis of the FrTp CPN is an iterative process. Simulation of the CPN is used to step through selected sequences to validate the operation of the model. Then, after defining a set of desired properties of the protocol, the state space is generated and the properties verified. As errors in the model are discovered the process is repeated. This section defines and reports the final set of desired properties and results from the state space and language analysis. These were in fact obtained from multiple iterations of simulations, analysis and refinement.

A. Desired Properties

A key property of any protocol is absence of deadlocks. Deadlocks are identified as unexpected terminal markings in the state space. A terminal marking is defined as (from Definition 9.19 in [6]):

Definition 1 (Terminal Marking)**.** *M is a terminal marking if and on if:*

$$\forall t \in T : \neg(M \xrightarrow{t}) \tag{1}$$

The set of terminal markings is M_{TM}. Three sets of expected terminal markings—M_{SS}, M_{UU} and M_{US}—are defined based on the intended operation of FrTp. For brevity assume:

$$P_{channel} = \{\text{Data\_Channel, Ack\_Channel}\} \tag{2}$$

$$P_{buffer} = \{\text{Sender\_FRTP, Receiver\_FRTP}\} \tag{3}$$

Marking M_{SS} should be reached if both Sender and Receiver have successfully completed the data transfer:

$$M_{SS}(p) = \begin{cases} Successful & \text{if } p = \text{To\_Tx\_PduR} \\ (0,0,x,x,\_) & \text{if } p = \text{Data\_to\_Send} \\ Indication & \text{if } p = \text{To\_Rx\_PduR} \\ true & \text{if } p = \text{Indication\_Sent} \\ (WD,WD,\_) & \text{if } p = \text{Data\_Received} \\ [] & \text{if } p \in P_{channel} \\ [] & \text{if } p \in P_{buffer} \\ \emptyset_{MS} & \text{if } p = \text{Confirmation} \\ \emptyset_{MS} & \text{if } p = \text{Ack\_Frame} \\ \_ & \text{otherwise} \end{cases} \tag{4}$$

$\emptyset_{MS}$ is the empty multiset (no tokens in the place), and following Standard ML notation _ is any value and [] is an empty list.

Marking M_{UU} should be reached if the data transfer is unsuccessful and both transmitter and receiver are aware of the failure:

$$M_{UU}(p) = \begin{cases} Unsuccessful & \text{if } p = \text{To\_Tx\_PduR} \\ \emptyset_{MS} & \text{if } p = \text{To\_Rx\_PduR} \\ false & \text{if } p = \text{Indication\_Sent} \\ [] & \text{if } p \in P_{channel} \\ [] & \text{if } p \in P_{buffer} \\ \emptyset_{MS} & \text{if } p = \text{Confirmation} \\ \emptyset_{MS} & \text{if } p = \text{Ack\_Frame} \\ \_ & \text{otherwise} \end{cases} \tag{5}$$

Finally, there may be a case where the receiver successfully receives the data, but the transmitter is not informed of this (e.g. the acknowledgement cannot be delivered successfully):

$$M_{US}(p) = \begin{cases} Unsuccessful & \text{if } p = \text{To\_Tx\_PduR} \\ Indication & \text{if } p = \text{To\_Rx\_PduR} \\ true & \text{if } p = \text{Indication\_Sent} \\ (WD, WD, \_) & \text{if } p = \text{Data\_Received} \\ [] & \text{if } p \in P_{channel} \\ [] & \text{if } p \in P_{buffer} \\ \emptyset_{MS} & \text{if } p = \text{Confirmation} \\ \emptyset_{MS} & \text{if } p = \text{Ack\_Frame} \\ \_ & \text{otherwise} \end{cases} \quad (6)$$

The first desired property of FrTp is:

Property 1 (Absence of deadlocks)**.** *The FrTp CPN contains no deadlocks if and only if:*

$$M_{TM} = M_{SS} \cup M_{UU} \cup M_{US} \quad (7)$$

Ideally the protocol language and desired service language should be equivalent. The desired service language ($\mathcal{L}_S$) is defined as the language accepted by the FSA in Fig. 6.

Property 2 (Language equivalence)**.** *The FrTp Protocol language ($\mathcal{L}_P$) is equivalent to the FrTp Service language ($\mathcal{L}_S$) if and only if:*

$$\mathcal{L}_P \subseteq \mathcal{L}_S \wedge \mathcal{L}_S \subseteq \mathcal{L}_P \quad (8)$$

Understanding the number of frames that can be in the network at any one time is useful in dimensioning transmit/receive buffers as well as the FlexRay bus capacity and utilisation. Although not explicitly stated in the specification [3], both the buffers and channels should be bounded. It is difficult to know the upper bounds prior to analysis, and hence state space analysis was used to determine the bounds. Therefore in this paper rather than defining the bounds as properties, we simply report the bounds measured from the state space analysis in the next section.

B. State Space Analysis

For a given set of input parameter values, CPN Tools can calculate the full state space of the FrTp CPN. Then CPN ML queries can be applied to prove properties from the state space. For example, a CPN ML query function was written to check if all terminal markings are one of either M_{SS}, M_{UU} or M_{US}.

The set of input parameter values analysed in this paper are[3]: WD: 100, 200, 300, ..., 1000 [bytes]; FS: 100 [bytes]; BS: 1, 2, 3; MR 0, 1, 2; $Loss$: true.

Results from the state space analysis are shown in Table I. The first three columns indicate input parameters. The next two columns give the size of the state space (States and Arcs). Columns six to nine give the number of markings in: M_{TM}; M_{SS}; M_{UU}; M_{US}. The last four columns give the maximum number of frames in: Data channel; Ack channel; Tx buffer; and Rx buffer.

The first observation from the state space analysis is that Property 1 (Absence of Deadlocks) is proved to hold for all configurations analysed (in Table I, $TM = SS + UU + US$).

Secondly, observing the measured upper bounds on data channel and buffer places, we conjecture that the buffer space needed for data frames is given by (9):

$$UB_{data} = \begin{cases} 1 + MR & \text{if } \frac{WD}{FS} \leq 2 \\ (BS - 1)(1 + MR) & \text{if } 2 < \frac{WD}{FS} \leq BS \\ BS(1 + MR) & \text{otherwise} \end{cases} \quad (9)$$

Equation 9 holds for the configurations analysed. In general, the maximum number of data frames in the buffer/channel at one time consists of the original blocks of CFs, plus retransmitted blocks of CF. Further work is needed to prove this for all cases.

State space analysis is powerful for investigating dynamic properties of the FrTp CPN. However there are two limitations of this analysis approach. Firstly, as the number of frames to be transmitted (WD) and the maximum retries allowed (MR) increase, the number of states increased. This is the state explosion problem, and eventually leads to state spaces that cannot be calculated in reasonable time/memory[4]. A promising technique to alleviate the state explosion problem is to utilise the sweep-line method [17]. This method discards states from memory if the can no longer be reached. This has been applied for other protocols, where properties can be proved for state spaces 2-3 times the size of when sweep-line is not applied [12]. In FrTp the number of retries could be used as a progress measure to determine which states can/cannot be reached in the future.

Another limitation is that a separate state space must be calculated for every combination of initial parameter values. For instance, FrTp supports block sizes of 1 to 16. The analysis has only so far considered block sizes of 1 to 3. As future work the CPN model can be modified to abstract from some of the initial parameters (e.g. not

[3]In fact other parameters values have also been used, e.g. with WD not a multiple of FS and with no frame loss, but are omitted due to space.

[4]The state space sizes shown in Table I are quite small; a configuration not reported had 100,000 nodes generated in 1 hour

TABLE I
STATE SPACE ANALYSIS OF FRTP PROTOCOL CPN

WD	BS	MR	S	A	TM	SS	UU	US	Data	Ack	Tx	Rx
100	1	0	36	67	3	1	1	1	1	1	1	1
200	1	0	64	119	5	1	3	1	1	1	1	1
300	1	0	92	171	7	1	5	1	1	1	1	1
400	1	0	120	223	9	1	7	1	1	1	1	1
500	1	0	148	275	11	1	9	1	1	1	1	1
600	1	0	176	327	13	1	11	1	1	1	1	1
700	1	0	204	379	15	1	13	1	1	1	1	1
800	1	0	232	431	17	1	15	1	1	1	1	1
900	1	0	260	483	19	1	17	1	1	1	1	1
1000	1	0	288	535	21	1	19	1	1	1	1	1
100	1	1	170	493	4	2	1	1	2	2	2	2
200	1	1	496	1442	8	2	5	1	2	2	2	2
300	1	1	790	2270	11	2	8	1	2	2	2	2
400	1	1	1096	3140	14	2	11	1	2	2	2	2
500	1	1	1402	4010	17	2	14	1	2	2	2	2
600	1	1	1708	4880	20	2	17	1	2	2	2	2
700	1	1	2014	5750	23	2	20	1	2	2	2	2
800	1	1	2320	6620	26	2	23	1	2	2	2	2
900	1	1	2626	7490	29	2	26	1	2	2	2	2
1000	1	1	2932	8360	32	2	29	1	2	2	2	2
100	1	2	550	2024	5	3	1	1	3	3	3	3
200	1	2	2996	11488	12	3	7	2	3	3	3	3
300	1	2	4861	18001	16	3	11	2	3	3	3	3
400	1	2	7093	26114	20	3	15	2	3	3	3	3
500	1	2	9361	34323	24	3	19	2	3	3	3	3
600	1	2	11629	42532	28	3	23	2	3	3	3	3
700	1	2	13897	50741	32	3	27	2	3	3	3	3
800	1	2	16165	58950	36	3	31	2	3	3	3	3
900	1	2	18433	67159	40	3	35	2	3	3	3	3
1000	1	2	20701	75368	44	3	39	2	3	3	3	3
100	2	0	36	67	3	1	1	1	1	1	1	1
200	2	0	64	119	5	1	3	1	1	1	1	1
300	2	0	115	246	6	1	4	1	2	1	2	2
400	2	0	143	298	8	1	6	1	2	1	2	2
500	2	0	194	425	9	1	7	1	2	1	2	2
600	2	0	222	477	11	1	9	1	2	1	2	2
700	2	0	273	604	12	1	10	1	2	1	2	2
800	2	0	301	656	14	1	12	1	2	1	2	2
900	2	0	352	783	15	1	13	1	2	1	2	2
1000	2	0	380	835	17	1	15	1	2	1	2	2
100	2	1	170	493	4	2	1	1	2	2	2	2
200	2	1	524	1545	8	2	5	1	2	2	2	2
300	2	1	1468	5082	10	2	7	1	4	2	4	4
400	2	1	3455	12446	13	2	10	1	4	3	4	4
500	2	1	5376	19924	15	2	12	1	4	4	4	4
600	2	1	7815	29150	18	2	15	1	4	4	4	4
700	2	1	10990	42127	20	2	17	1	4	4	4	4
800	2	1	13715	52301	23	2	20	1	4	4	4	4
900	2	1	17037	65853	25	2	22	1	4	4	4	4
1000	2	1	19852	76309	28	2	25	1	4	4	4	4
100	3	0	36	67	3	1	1	1	1	1	1	1
200	3	0	64	119	5	1	3	1	1	1	1	1
300	3	0	115	246	6	1	4	1	2	1	2	2
400	3	0	281	749	7	1	5	1	3	2	3	3
500	3	0	309	801	9	1	7	1	3	2	3	3
600	3	0	360	928	10	1	8	1	3	2	3	3
700	3	0	526	1431	11	1	9	1	3	2	3	3
800	3	0	554	1483	13	1	11	1	3	2	3	3
900	3	0	605	1610	14	1	12	1	3	2	3	3
1000	3	0	771	2113	15	1	13	1	3	2	3	3

modelling the data size, WD, explicitly). For analysis, approaches such as [18] that verify properties independent of the initial parameters can also be considered.

C. Language Analysis

The FrTp protocol language, $\mathcal{L}_P$, is the set of interactions possible between FrTp and the PDU Router. Property 2 requires it to be equivalent to the service language in Fig. 6. There are four transitions in the FrTp CPN that correspond to interactions between FrTp and PDU Router: FrTp_Transmit, FrTp_Indication, Successful and Unsuccessful (the last two, on page PduR_FrTpTxConfirmation, are cases of the primitive Confirmation). All binding elements (arcs) in the state space that contain these four transitions are mapped to their corresponding primitive. All other binding elements (arcs) in the state space are considered ϵ-transitions. As a result the state space can be treated as a non-deterministic FSA, which can be reduced to its canonical form, a minimised deterministic FSA, representing the protocol language.

For every configuration considered in Section V-B, the protocol language produced was equivalent to the service language (hence proving Property 2).

VI. Conclusions

AUTOSAR is an architecture for developing and deploying embedded applications in vehicles. One of the many components of AUTOSAR is the FlexRay Transport Protocol, which provides additional reliability and efficiency to the FlexRay inter-ECU communications bus. Key contributions of the paper are:

1) A formal Coloured Petri net model of the AUTOSAR FlexRay Transport Protocol.
2) Definition of the desired FrTp service language.
3) Definition of the desired properties of the FrTp protocol, in particular the expected terminal states.
4) Proof, via state space and language analysis, that the FrTp protocol is free of deadlocks and faithfully refines the FrTp service, for a given set of initial parameter values.
5) Characterisation of the upper bounds of the FrTp transmit and receive buffers.

Future work includes verifying other configurations of FrTp, and modifying the model to alleviate the problems of using state space analysis (state explosion, dependence on initial parameters). Once functional verification is achieved, the model can be adapted to support performance evaluation, completing the analysis and verification of the FlexRay Transport Protocol.

References

[1] FlexRay Consortium, "FlexRay Protocol Specification," V2.1 Rev A. Available via: http://www.flexray.com/, 12 Dec. 2000.

[2] AUTOSAR, "Technical Overview," V2.2.2 R3.1. Available via: http://www.autosar.org, Aug. 2008.

[3] ——, "Specification of FlexRay Transport Layer," V2.2.2 R3.1. Available via: http://www.autosar.org, Aug. 2008.

[4] G. J. Holzmann, *Design and Validation of Computer Protocols*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[5] J. Billington, G. E. Gallasch, and B. Han, "A Coloured Petri net approach to protocol verification," in *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*. Springer-Verlag, 2004, pp. 210–290.

[6] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Berlin: Springer, 2009.

[7] OSEK/VDX, "OSEK Communications Specification," V3.0.3. Available via: http://www.osek-vdx.org, Jul. 2004.

[8] ISO, "Road Vehicles—Diagnostics on Controller Area Networks (CAN)—Part 2: Network layer services," ISO 15765-2, 2004.

[9] S. Vanit-Anunchai, "Towards formal modelling and analysis of SCTP connection management," in *Proc. 9th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, 20-22 Oct. 2008.

[10] L. Liu and J. Billington, "Verification of the Capability Exchange Signalling protocol," *Intl. Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 305–326, Jun. 2007.

[11] G. E. Gallasch and J. Billington, "Parametric language analysis of the class of stop-and-wait protocols," in *Proc. 29th Intl. Conf. Application and Theory of Petri Nets and Other Models of Concurrency*, Xi'an, China, 25-27 Jun. 2008.

[12] S. Gordon, L. M. Kristensen, and J. Billington, "Verification of a revised WAP Wireless Transaction Protocol," in *Proc. Intl. Conf. on Application and Theory of Petri Nets*, Adelaide, Australia: Springer-Verlag, 24–30 Jun. 2002, pp. 182–202.

[13] J. Grossmann, D. Serbanescu, and I. Schieferdecker, "Testing embedded real time systems with TTCN-3," in *Proc. Intl. Conf. Software Testing, Verification, and Validation*, Denver, CO, 1–4 Apr. 2009, pp. 81–91.

[14] H. Moon, G. Kim, Y. Kim, S. Shin, K. Kim, and S. Im, "Automation test method for automotive embedded software based on AUTOSAR," in *Proc. Intl. Conf. on Software Engineering Advances*, Sep. 2009, pp. 158–162.

[15] M. Hillenbrand and K. D. Muller-Glaser, "An approach to supply simulations of the functional environment of ECUs for hardware-in-the-loop test systems based on EE-architectures conform to AUTOSAR," in *Proc. Intl. Work. Rapid System Prototyping*, Paris, France, 23–26 Jun. 2009, pp. 188–195.

[16] O. Kallel, S. Dridi, and S. Hasnaoui, "Modeling and evaluating a CAN controller components using stochastic and Colored Petri nets," *Intl. Review on Computers and Software*, vol. 4, no. 1, pp. 142–151, Jan. 2009.

[17] S. Christensen, L. M. Kristensen, and T. Mailund, "A sweep-line method for state space exploration," in *Proc. Intl. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, Genova, Italy: Springer-Verlag, 2-6 Apr. 2001, pp. 450–464.

[18] G. E. Gallasch and J. Billington, "A parametric state space for the analysis of the infinite class of stop-and-wait protocols," in *Proc. Intl. SPIN Workshop on Model Checking of Software*, Vienna, Austria, 30 March - 1 April 2006, pp. 201–218.