Transforming State Tables to Coloured Petri Nets for Automatic Verification of Internet Protocols

San Choosang and Steven Gordon
Sirindhorn International Institute of Technology
Thammasat University
Bangkadi, Thailand 12000
Email: schoosang@ict.siit.tu.ac.th, steve@siit.tu.ac.th

Abstract— **Rapid developments in networking technologies is resulting in an increasing number of new communication protocols being created, but formal methods are seldom used to verify their design. This paper presents a set of rules for transforming state tables, a common format of protocol specifications in standards, into a formal model based on Coloured Petri nets. This reduces time for developing and debugging CPN models, which can then be used for protocol verification. Formal definitions of subsets of state tables and CPNs are presented, as well as the transformation algorithm. To demonstrate the transformation an example of Stop-and-Wait protocol is used as a case study.**

Index Terms—**protocol verification, Coloured Petri nets, XML, Stop-and-Wait protocol**

I. Introduction

Nowadays many new communication protocols have been created to improve the capability and the performance of networking technologies. It is important that the design of the protocol is proved to be free of significant errors to ensure that the protocol operates correctly without undesired or unsafe behavior. Formal methods are well-suited to protocol design activities [1]; they can increase confidence that the design is free of errors that would be expensive to fix once a protocol is deployed into the network. However due to the cost of applying formal methods (steep learning curve and time consuming), no upcoming standards apply them for protocol verification. This research aims to bridge this gap by automating the task of producing an executable formal model of common protocols.

State (transition) tables are a common method used in standards to specify a protocol [2], [3], [4]. However they have limitations, in particular they lack tools and techniques that allow automatic proof of properties relevant to a protocol (e.g. absence of deadlocks and livelocks). Coloured Petri nets (CPNs) [5], a formal modelling language with a graphical notation, do have such support via model checking techniques [6]. However developing and debugging CPN models of protocols is a time consuming process, taking days to weeks for experienced users. The motivation of our research is to reduce the development time, so a protocol designer can integrate formal methods such as CPNs into their workflow. Therefore this paper contributes a novel approach for automatically converting a state table protocol specification into a CPN model.

As far as we know, although several researchers have developed CPNs manually based on state tables [7], no attempts have been made to automatically convert state tables to CPNs. Section II presents other work that has applied similar transformations. Our research assumes protocols of a specific type. To support the transformation, in Section III we describe the assumptions and contribute a new definition of both state tables and CPNs that fits the assumptions. In Section IV we present our proposed transformation algorithm, taking a state table as input and producing a CPN as output. We also describe our XML/XSLT-based implementation of the transformation. To demonstrate the transformation we apply it to an example Stop-and-Wait protocol, with results presented in Section V. Concluding remarks are given in Section VI.

II. Related Work

A simple way to present the behavior of the protocol in the specification is using a graphical notation, an informal language i.e. UML, which is not designed for the protocol verification purpose. [8] proposes an approach to transform an UML statechart and collaboration diagram to CPN model by using graph grammars and graph transformation techniques, while [9] implements an automated tool that can transform a Live Sequence Chart (LSC) to CPN models. This tool reads the LSC model as an input and transform the system's behavior into a unified CPN model. The motivation of these two research works is similar to ours; reducing the CPN model development time. However they do not handle state tables as input, which are common in protocol specifications [2], [3], [4]. Our transformation approach is inspired by the work in [10], [11], which transforms descriptions of railway interlocking tables into CPNs using XML and XSLT.

III. State Tables and CPNs

State tables are a common format for designers to specify protocols in standards. In Section III-A we give a brief description of state tables. Although state tables can be treated as a finite state automata (FSA) and are subject for formal analysis, in practice they are developed in an informal manner with various different formats. Benefits of converting state tables to CPNs include utilising the various Petri net theory and software for simulation and formal analysis. Section III-B describes CPNs, while Section III-C explains a common approach for modelling protocols with CPNs. To illustrate concepts, an example Stop-and-Wait protocol is used in this paper.

A. State Tables

A state table contains a set of states, events, conditions, and actions. In state s if event e occurs and conditions c_1, c_2, $\dots c_n$ are true, then actions a_1, a_2, $\dots a_m$ are taken and the next state is entered. If we specifically consider a communication protocol with two entities (sender and receiver), each entity has state tables for a particular state of that entity. In a protocol the set of events can be classified as those relating to receiving packets from lower layer, receiving packets from higher layer, or timeouts occurring. Similarly, actions maybe: transmit a packet, change the value of a timer, increment/decrement a counter, or set the value of a variable. We have expressed these classifications in a formal definition of a protocol state table in Figure 1. As an example, Figure 2 shows the state tables for a Stop-and-Wait protocol. Consider the sender in the *IDLE* state. The event *rxHL_Msg* is in the set E_{rxHL}. There is one action in the set A_{tx} (tx_Data) and one in A_{timer} (timer_toRTx_start).

1) $S = \{s \mid \text{s is a state in protocol specification}\}$ is a finite set of **states**.
2) $E = \{E_{RxLL}, E_{RxHL}, E_{To}\}$ is a finite set of **events**, where:
   - E_{RxLL} is an event relating to receiving packets from lower layer.
   - E_{RxHL} is an event relating to receiving packets from high layer.
   - E_{To} is an event relating to timeout occurring.
3) C = is a finite set of expression such that *Type[C] = Bool* and is called **conditions**.
4) $A = \{A_{tx}, A_{timer}, A_{increment}, A_{decrement}, A_{setvalue}\}$ is a finite set of **actions**, where:
   - A_{tx} is an action that transmits a packet to a communication channel.
   - A_{timer} is an action that changes the value of a timer variable.
   - $A_{increment}$ is an action that increases the value of a counter variable by one.
   - $A_{decrement}$ is an action that decreases the value of a counter variable by one.
   - $A_{setvalue}$ is an action that sets the value of a variable.
5) $\Delta : (S \times E \times C) \rightarrow A$ is an **action function** that assigns a set of (A) to each combination of $(s \times e \times c)$ such that $s \in S$, $e \in E$, and $c \in P(C)$.
6) $NS : (S \times E \times C) \rightarrow S$ is an **next state function** that assign a next stage to each combination of $(s \times e \times c)$ such that $s \in S$, $e \in E$, and $c \in P(C)$.

Figure 1. Formal Definition of Protocol State Table

B. CPNs

CPNs are a directed graph with two types of nodes: a set of places, P, and a set of transitions, T, represented by ellipses and rectangles, respectively. Places and transitions are connected by directed arcs: input arcs (place to transition) and output arcs (transition to place). Places are typed by a colour set and the values that marks on the places are called tokens. Transitions and arcs can also have inscriptions (expressions) to control the execution of the model. The execution of a CPN consists of occurrence of transitions. A transition can occur if and only if: for all input places, sufficient tokens exist that satisfy the input arc inscriptions, and the transition inscription

SENDER SIDE

State : IDLE

Event	Condition	Action	Next State
rxHL_Msg	Finish = false	tx_Data timer_toRTx_start	WAIT
rxLL_Ack	–	tx_Data	IDLE
rxLL_Nack	–	–	IDLE

State : WAIT

Event	Condition	Action	Next State
rxLL_Ack	–	timer_toRTx_stop set_Finish_true	IDLE
rxLL_Nack	Retry < MR	tx_Data timer_toRTx_restart ct_Retry_inc	WAIT
rxLL_Nack	Retry >= MR	timer_toRTx_stop set_Finish_false	IDLE
to_RTx	Retry < MR	tx_Data timer_toRTx_restart ct_Retry_inc	WAIT
to_RTx	Retry >= MR	timer_toRTx_stop set_Finish_false	IDLE

RECEIVER SIDE

State : WAIT

Event	Condition	Action	Next State
rxLL_CorrectData	–	tx_Ack set_Finish_true	WAIT
rxLL_!CorrectData	–	tx_Nack set_Finish_false	WAIT

Figure 2. State Tables of Stop-and-Wait Protocol

evaluates to true. A formal definition of CPNs is presented in [5].

C. Modelling Protocol with CPNs

There are many ways to model a protocol with CPNs [5], [6], depending on the objectives of the modeller. In our work we consider the objectives of producing a CPN model that is structured similar to the state table description (for validation), and that is amenable to state space analysis (for verification). Hence a state-based approach for modelling is used [7]. This is described via a general example as illustrated in Figure 3. The modelling approach assumes a unicast protocol with two entities, sender and receiver, communicating by a single full-duplex channel. The current state of each entity, and associated state variables are stored in a single place, p_{Sender} and $p_{Receiver}$. Each event is modelled by a single transition, when an event occurs the state's information is updated by A_{sndCS} (containing the current state name and state variables) and A_{sndNS} (containing the next state name and actions to update the state variables) for the p_{Sender} place and vice versa, A_{rcvCS} and A_{rcvNS}, for the $p_{Receiver}$ place. To transmit and receive packets from the communication channels, p_{S2R} and p_{R2S}, four arcs are used; A_{sndTx}, A_{sndRx}, A_{rcvTx}, and A_{rcvRx}. For the communication channels we assume that there is no packet loss, and the timer events are considered nondeterministic (it either may occur or may not occur at any time).

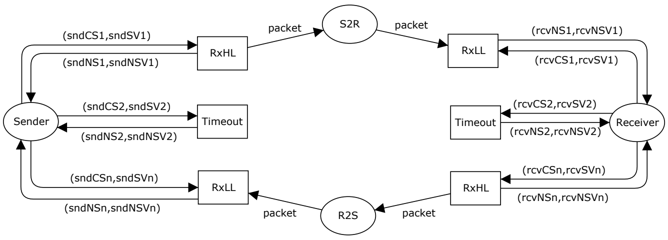

Figure 3. Overall Model of a Refined Definition

We have formalised this subset of CPNs for modelling protocols to a protocol CPNs definition in Figure 4.

IV. Automatically Generating a Protocol CPN

A. Approach

Our objective is to allow a protocol designer to manually create a state table specification of a protocol, then automatically generate the CPN model from the state tables, after which analysis of the CPN state space can be performed (again, automatically). This saves time in developing the CPN and hides many of the complexities of CPNs from the designer.

We assume the protocol designer can create the necessary state tables. However to support conversion to CPNs, a predefined syntax must be used. The syntax should capture all parts of the state table definition in Figure 1. A graphical editor could be used to aid the designer in following the syntax. For our prototype we use the syntax as illustrated in the example Figure 2. For events we use underscore (_) to separate between event's type (e.g. RxLL, RxHL, To) and event subject (e.g. packet's type or timer variable). For actions, we have three portions separated by underscore; action's type, target variable, and value, respectively (excepting the transmit action that has only two portions, action's type and packet's type).

Once the state table exists, it must be transformed to a CPN. We present the transformation rules in Figure 5. These rules take any protocol matching the state table definition in Figure 1, and produces a protocol matching the CPN definition in Figure 4. In the rules we use dot (.) to distinguish the element between the two definitions, for example $st.E_{RxLL}$ is an element in state table definition.

Consider at the sender, each row in the state table represents an event. Hence for each row a CPN transition is created to model that event. Arcs are created between the transition and the place *Sender*. The arc $cpn.A_{sndCS}$ contains an inscription specifying the current state name and variables. The state name and variables on this arc means a condition of the transition being enabled is that the sender is in the named state with the appropriate variables. The arc $cpn.A_{sndNS}$ contains an inscription specifying the next state name and actions to update the variables.

In addition, if the event is related to receiving a packet from the lower layer, an additional $cpn.A_{sndRx}$ arc is created from the place *R2S* to this transition, i.e. a condition for the

1) $P = \{p_{Sender}, p_{Receiver}, p_{S2R}, p_{R2S}\}$
2) $T = \{T_{RxLL}, T_{RxHL}, T_{To}\}$, where:
   - T_{RxLL} is a finite set of **RxLL transitions** T_{RxLL} that receive a packet from lower layer.
   - T_{RxHL} is a finite set of **RxHL transitions** T_{RxHL} that receive a packet from higher layer.
   - T_{To} is a finite set of **To transitions** T_{To} that involve the timeout occurring.
3) $A = \{A_{sndCS}, A_{sndNS}, A_{rcvCS}, A_{rcvNS}, A_{sndTx}, A_{sndRx}, A_{rcvTx}, A_{rcvRx}\}$, where:
   - $A_{sndCS} \subseteq p_{Sender} \times T$ is a set of **sndCS arcs**.
   - $A_{srcNS} \subseteq T \times p_{Sender}$ is a set of **sndNS arcs**.
   - The definition of A_{rcvCS}, A_{rcvNS} follow the same style as A_{sndCS} and A_{sndNS}, respectively.
   - $A_{sndTx} \subseteq T \times p_{S2R}$ is a set of **sndTx arcs**.
   - $A_{sndRx} \subseteq p_{R2S} \times T$ is a set of **sndRx arcs**.
   - The definition of A_{rcvTx}, A_{rcvRx} follow the same style as A_{sndTx} and A_{sndRx}, respectively.
4) $\Sigma = \{StateInfo, StateName, StateVar, Packet, INT, BOOL, STRING, UNIT\}$
5) $V = \{V_{sndCS}, V_{sndNS}, V_{rcvCS}, V_{rcvNS}, V_{sndSV}, V_{sndNSV}, V_{rcvSV}, V_{rcvNSV}, V_{packet}\}$, where:
   - V_{sndCS} is a set of **sndCS variables** such that $Type[v] \in StateName$ for all variables $\forall v \in V_{sndCS}$.
   - The definition of V_{sndNS}, V_{rcvCS} and V_{rcvNS} follow the same style as V_{sndCS}.
   - V_{sndSV} is a set of **sndSV variables** such that $Type[v] \in StateVar$ for all variables $\forall v \in V_{sndSV}$.
   - The definition of V_{sndNSV}, V_{rcvSV} and V_{rcvNSV} follow the same style as V_{sndSV}.
   - V_{packet} is a set of **packet variables** such that $Type[v] \in Packet$ for all variables $\forall v \in V_{packet}$.
6) $C : P \to \Sigma$ is a **colour set function** that assigns a colour set to each place.
$$C(p) = \begin{cases} StateInfo & \text{if } p \in \{p_{Sender}, p_{Receiver}\} \\ Packet & \text{if } p \in \{p_{S2R}, p_{R2S}\} \end{cases} \quad (1)$$
7) $G : T \to EXPR_V$ is a **guard function** that assigns a guard to each transition t such that $Type[G(t)] = Boolean$.
8) $E = \{E_{sndCS}, E_{sndNS}, E_{rcvCS}, E_{rcvNS}, E_{sndTx}, E_{sndRx}, E_{rcvTx}, E_{rcvRx}\}$, where:
   - $E_{sndCS} : A_{sndCS} \to (V_{sndCS}, V_{sndSV})$ is an **sndCS arc expression function** that assigns an arc expression to each arc $a_{A_{sndCS}}$ such that $Type[E(a_{A_{sndCS}})] = C(p_{Sender})_{MS}$.
   - $E_{sndNS} : A_{sndNS} \to (V_{sndNS}, V_{sndNSV})$ is an **sndNS arc expression function** that assigns an arc expression to each arc $a_{A_{sndNS}}$ such that $Type[E(a_{A_{sndNS}})] = C(p_{Sender})_{MS}$.
   - The definition of E_{rcvCS} and E_{rcvNS} follow the same style as E_{sndCS} and E_{sndNS}, respectively.
   - $E_{sndTx} : A_{sndTx} \to (V_{packet})$ is an **sndTx arc expression function** that assigns an arc expression to each arc $a_{A_{sndTx}}$ such that $Type[E(a_{A_{sndTx}})] = C(p_{S2R})_{MS}$.
   - The definition of E_{sndRx}, E_{rcvTx} and E_{rcvRx} follow the same style as E_{sndTx}.
9) $I : P \to EXPR_{\emptyset}$ is an **initialisation function** that assigns an initialisation expression to each place p such that $Type[I(p)] = C(p)_{MS}$.

Figure 4. Formal Definition of Protocol CPNs

transition being enabled is that a token representing a packet is in the place *R2S*.

If the event type is a timeout, then a guard is added to the transition, i.e. a condition for the transition being enabled is that the timer has started. For all types of events, any conditions in the state table are included in the transition guard.

Each action types namely, A_{timer}, $A_{increment}$, $A_{decrement}$, and $A_{setvalue}$ are included in the inscription of $cpn.A_{sndNS}$ arc to update the variables of that event when next state is reached.

If the action type is an action that transmits a packet to the communication channel, an additional $cpn.A_{sndTx}$ arc is created from this transition to the place *S2R*.

The receiver transformation is similar to that of the sender.

Rules for sender side state tables

foreach row in state tables
  create $cpn.T$ transition for $st.E$ event
  if $(e \in st.E_{RxLL})$
    create an $cpn.A_{sndRx}$ arc with $cpn.E_{sndRx}$ inscription
  create an $cpn.A_{sndCS}$ arc with $cpn.E_{sndCS}$ inscription
  create an $cpn.A_{sndNS}$ arc with $cpn.E_{sndNS}$ inscription
  if $(e \in st.E_{To})$
    $cpn.G(T) =$ timer variable of e is *true*
  $cpn.G(T) = st.C$
  foreach Actions $st.A$
    if $(a \in st.A_{tx})$
      create $cpn.A_{sndTx}$ arc with $cpn.E_{sndTx}$ inscription
    else
      Update $cpn.V_{sndNSV}$ variables in $cpn.E_{sndNS}$ inscription with $st.A$ action
  end foreach
  Update $cpn.V_{sndNS}$ variables in $cpn.E_{sndNS}$ inscription with $st.S$ NextState
end foreach

Rules for receiver side state tables

foreach row in state tables
  create $cpn.T$ transition for $st.E$ event
  if $(e \in st.E_{RxLL})$
    create an $cpn.A_{rcvRx}$ arc with $cpn.E_{rcvRx}$ inscription
  create an $cpn.A_{rcvCS}$ arc with $cpn.E_{rcvCS}$ inscription
  create an $cpn.A_{rcvNS}$ arc with $cpn.E_{rcvNS}$ inscription
  if $(e \in st.E_{To})$
    $cpn.G(T) =$ timer variable of e is *true*
  $cpn.G(T) = st.C$
  foreach Actions $st.A$
    if $(a \in st.A_{tx})$
      create $cpn.A_{rcvTx}$ arc with $cpn.E_{rcvTx}$ inscription
    else
      Update the $cpn.V_{rcvNSV}$ variables in $cpn.E_{rcvNS}$ inscription with $st.A$ action
  end foreach
  Update the $cpn.V_{rcvNS}$ variables in $cpn.E_{rcvNS}$ inscription with $st.S$ NextState
end foreach

Figure 5. State Table to CPNs Model Transformation Rules

B. Implementation

We have implemented the transformation using XSLT [12] by taking an XML state table as an input and producing a CPN model file that can be loaded in CPN Tools as an output (as illustrated in Figure 6). The XML state table is the representation of the protocol state tables in XML format. Since this is an initial prototype we assume that the XML state table already exists. A parsing tool can help the protocol designer to parse the contents in the state tables into an XML format.

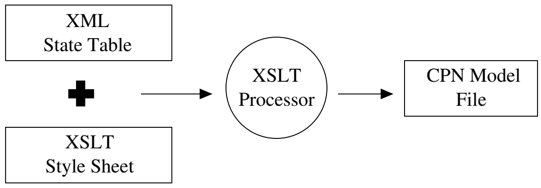

Figure 6. Transformation Process

The XSLT style sheet is written follow the transformation rules in Figure 5, it will read through the XML state table and create an appropriate CPN model element (*places* and *colour sets* are created follow 1) and 6) in Figure 4 before applying the transformation rules). The transformation process is automatically done by the *XSLT processor*. After loading the output file into CPN Tools, minor manual edits may be needed by the protocol designer. CPN Tools has in-built techniques for automating the protocol verification (e.g. state space analysis).

V. A Case Study: Stop-and-Wait Protocol

To demonstrate the transformation, we apply our approach to the Stop-and-Wait protocol (SWP) [13]. The state tables of SWP are shown in Figure 2 and some portion of XML state table of SWP, an input of the transformation, is shown in Figure 7 (this figure shows only the *IDLE* state of the sender side while the other parts are omitted). After the transformation process finished, the CPN model file of SWP is generated as an output. Some minor manual edits of the protocol model in CPN Tools are needed to complete the model, e.g. rearrange the model's elements for more readability or edit the arc inscription of the complex action function (none for this case study). The final SWP CPN model is shown in Figure 8.

```
<StateTables>
  <Sender>
    <State name="IDLE">
      <Event name="rx_HL_Msg">
        <Conditions>
          <Condition>Finish = false </Condition>
        </Condition>
        <Actions>
          <Action>tx_Data </Action>
          <Action>timer_RTx_start </Action>
        </Actions>
        <NextState name="WAIT"/>
      </Event>
      <Event name="rxLL_Ack">
        ...
      </Event>
      <Event name="rxLL_Nack">
        ...
      </Event>
    </State>
  </Sender>
  <Receiver>
    <State name="WAIT">
      ...
    </State>
  </Receiver>
</StateTables>
```

Figure 7. XML State Tables of Stop-and-Wait Protocol

Applying our transformation rules and prototype implemen-

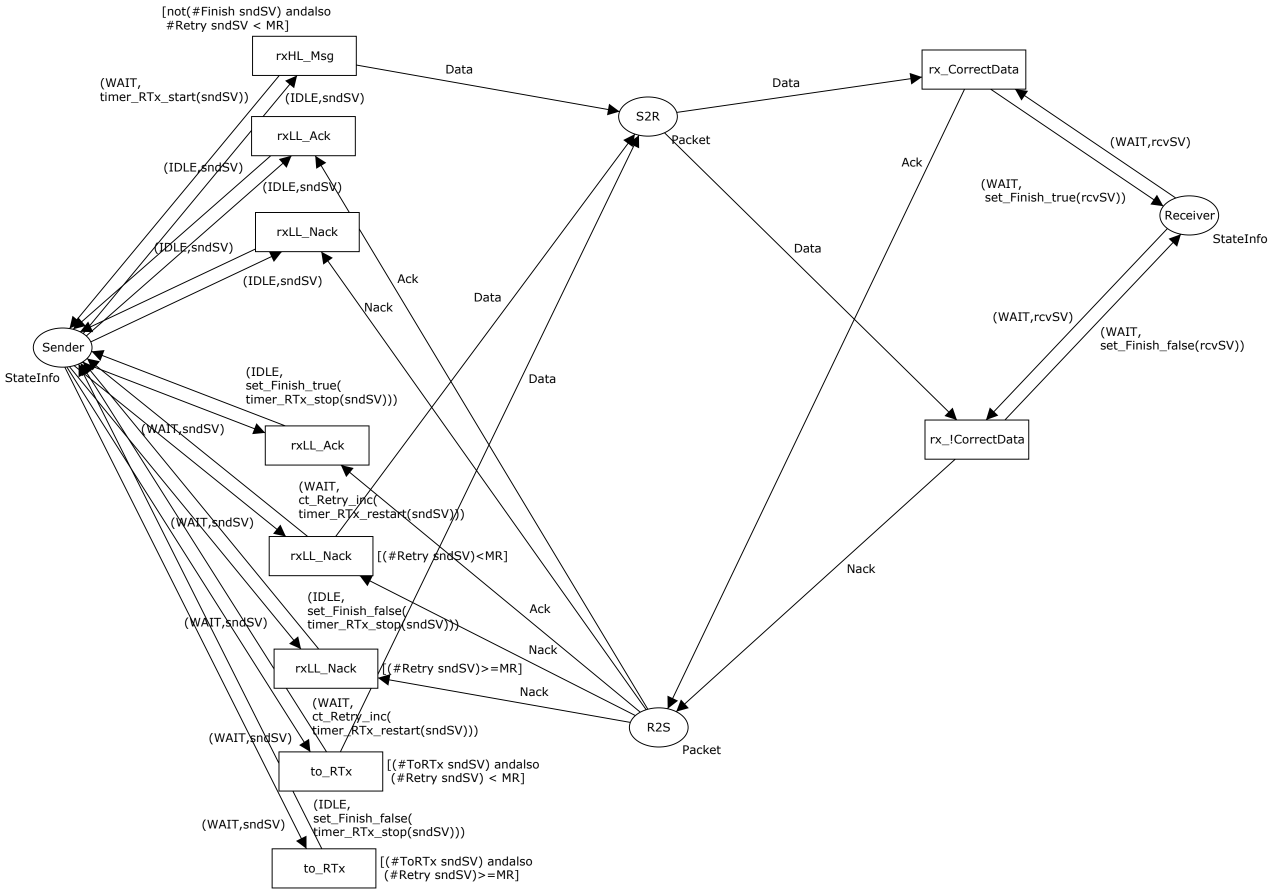

Figure 8. Stop-and-Wait CPNs Model From the Automatic Transformation

tation to a simple protocol have been successful. However there are a number of design challenges that exist.

1) Protocol's Variables and Constants: To declare the protocol's variables and constants automatically in this implementation, we add an additional section, declaration, in the XML state table to keep the protocol's variables (with type) and constants (with value). When the XSLT style sheet read through this section it will declare these variables and constants together with the common action functions for each variables in the CPN model. This is not the best approach for the automatic declaration but it is simple and suited for the first prototype of the implementation. Other approach that can be used for the automatic declaration is writing the XSLT that can extract the necessary informations, i.e. variable or constant name, from the XML state table by itself.

2) Manually Editing: One issue of the implementation is we need to manually edit some arc inscriptions, E_{sndNS} and E_{rcvNS}, which have complex action functions. As stated early this implementation can support (automatically generate) only the common action functions such as increment counter, start/stop timer. For more complex functions, such as create a special packet to send, the user must declare the function manually in the CPN model.

3) Model Readability: The output model of this implementation is generated in one page. If the input protocol has a lot of states and events, the output model will lack readability since there will be many transitions in one page. To overcome this issue in the future, we can group the events in one state table and put them into a subpage of the model. For example in Figure 2 consider the *IDLE* state of sender, we can put the transitions that represent *rxHL_Msg*, *rxLL_Ack*, and *rxLL_Nack* into a subpage under substitution transition named *IDLE*.

4) Reverse Transformation: By using this approach to transform state tables into CPN model we can modify the transformation rules to achieve the reverse transformation, producing the state tables from the existing CPN model. This will be useful for including state tables, verified from the corresponding CPN model, directly into standards.

VI. Conclusion and Future Work

The purpose of this research is to reduce the time to create formal CPN models of protocols, making it easier for protocol designers to identify design errors. In this paper we have described the process for transforming state tables specifications of protocols to CPN models. The key contributions are:

1) Refined formal definitions of state tables and CPNs specifically for unicast, two-entity protocols

2) An algorithm for transforming a state table to CPN
3) An implementation for the algorithm, applied to a Stop-and-Wait protocol.

Note that CPNs is one of the verification method that we have focused on this paper. To apply this transformation technique to the other verification method (e.g. SPIN/Promela), the algorithm for transforming is needed to modify to suit with target verification method.

Future work includes improving the capability of the transformation e.g. can handle many types of protocol not limited to the unicast protocol with two entities. Further evaluation of the transformation will be performed using a larger set of case studies, including protocols that already have a state table and CPN representation, as well as new protocols.

REFERENCES

[1] F. Babich and L. Deotto, "Formal methods for specification and analysis of communication protocols," *IEEE Communications Surveys & Tutorials*, vol. 4, no. 1, pp. 2–20, First quarter 2002.

[2] J. Vollbrecht, P. Eronen, N.Petroni, and Y. Ohba, *State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator*, IETF RFC 4137, August 2005.

[3] V. Fajardo, Y. Ohba, and R. Marin-Lopez, *State Machines for the Protocol for Carrying Authentication for Network Access*, IETF RFC 5609, August 2009.

[4] T. Tsenov, H. Tschofenig, X. Fu, C. Aoun, and E. Davies, *General Internet Signaling Transport (GIST) State Machine*, IETF RFC 5972, October 2010.

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

[6] 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.

[7] S. Gordon, "Towards verification of the pana authentication and authorisation protocol using coloured petri nets," in *Proceedings of the 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, October 2009, pp. 61–80.

[8] E. Kerkouche, A. Chaoui, O. Labbani, and E. Bourennane, "A uml and colored petri nets integrated modeling and analysis approach using graph transformation," *Object Technology*, vol. 9, no. 4, pp. 25–43, 2010.

[9] B. Khadka, "Transformation of live sequence charts to colored petri nets (lsctocpn)," Ph.D. dissertation, University of Massachusetts Dartmouth, January 2007.

[10] S. Vanit-Anunchai, "Verification of railway interlocking tables using coloured petri nets," in *Proceedings of the 10th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Aarhus, Denmark, October 2009, pp. 139–158.

[11] S. Vanit-Anunchai , "Modelling railway interlocking tables using coloured petri nets," in *Proceedings of the 12th International Conference on Coordination Models and Languages*, Volume 6116 of Lecture Notes in Computer Science. Amsterdam, The Netherlands: Springer-Verlag, June 2010, pp. 137–151.

[12] S. Muench, *Building Oracle XML Applications*. O'Reilly Media, 2000, ch. 7, pp. 275–309.

[13] W. Stallings, *Data and Computer Communications*, 8th ed. Prentice Hall, September 2010.