A Coloured Petri Net Approach to Formalising and Analysing the Resource Reservation Protocol

María E. Villapol

School of Computer Sciences
Central University of Venezuela
Av. Los Ilustres, Los Chaguaramos, Caracas, Venezuela
Tel: +212 58 605 1132, Fax: +212 58 605 1131
Email: mvillap@strix.ciens.ucv.ve

and

Jonathan Billington

Computer Systems Engineering Centre
University of South Australia
Mawson Lakes, Adelaide, SA, 5095, Australia
Tel: +618 8302 3940, Fax: +618 8302 3384
Email: j.billington@unisa.edu.au

Abstract

The goal of the *Resource Reservation Protocol* (RSVP) is to support the provision of the *Quality of Service* required for emerging Internet applications (such as video conferencing) that require a level of performance not guaranteed by the Internet. RSVP attempts to provide performance guarantees by establishing resource reservations (such as the number of buffers and bandwidth allocation) within routers and host computers of the Internet. Currently, Internet protocols are not formally specified when they are developed. Instead they are described in a narrative way in documents called *Request for Comments* (RFCs). This is the case for RSVP. To increase confidence in RSVP we have formalised and analysed its narrative specification using *Coloured Petri Nets* (CPNs). This paper demonstrates how CPNs can be used for modelling and analysing RSVP. Among the several beneficial features of CPNs are: graphical facilities for specification; support for different levels of abstraction; hierarchical structuring mechanisms; and verification and validation methods, such as querying the state space to investigate properties, and language equivalence to check the consistency of different levels of abstraction. Coloured Petri Nets are supported by a number of computer tools including Design/CPN. Design/CPN supports the construction and maintenance of CPN models and their simulation and analysis using state spaces. These facilities allow us to create a model that provides a clear, unambiguous and precise definition of RSVP, and to analyse the protocol for functional correctness. The paper concentrates on the approach and the tools used in this investigation.

Keywords: RSVP, Quality of Service (QoS), Coloured Petri Nets, State Spaces.

1 Introduction

The *Internet Integrated Service Model (IntServ)* [7] is one of the proposals for providing the desired *Quality of Service (QoS)* for applications operating over the Internet. QoS guarantees are required for multimedia and real-time applications. The *Resource Reservation Protocol (RSVP)* [8][9] is part of the IntServ model. RSVP is a signalling protocol developed to create and maintain resource reservations (eg buffer and data rate allocations) in Internet routers and host computers, to provide the desired QoS. RSVP uses a soft-state approach, where state information concerning traffic characteristics and resource reservations must be periodically refreshed, or else face automatic removal. For the desired QoS to be guaranteed it is essential that RSVP works correctly.

Formal methods provide techniques to support the design, verification and testing of communication protocols [1][2][23]. We have found that formal techniques are only recently being applied to Internet protocols. Some of this work has focused on TCP [4][5][6][10][11][15][24], the Datagram Congestion Control Protocol (DCCP) [27], the Internet Open Trading Protocol (IOTP) [18][19][20][21] and RSVP (which includes our own work [28][29][30][31]). *Coloured Petri Nets (CPNs)* [14] are a formal technique used for modelling many systems, particularly communication protocols [3]. The work presented here is based on an extensive investigation of RSVP mechanisms using CPNs [30]. Our first attempt at modelling and analysis of RSVP was presented in [28]. We then developed a service specification for RSVP [29]. Since then, the model has been significantly revised, restructured and refined to include more features (e.g. confirmation of reservations). The new model has been developed using an incremental modelling and validation methodology [30] to increase our confidence in its validity. In [31], the CPN model of RSVP is analysed based on general properties of the protocol and a set of eight specific properties defined and formalised for the first time [31]. Part of the contribution of our work has been a clear, unambiguous and precise definition of the major features of the protocol, which is missing in the current specification of RSVP [8]. In contrast, this paper introduces our protocol verification methodology and CPNs in a relatively tutorial manner using an important Internet QoS signalling protocol (RSVP). We aim to provide an introduction to the way CPNs can be used to create a model of protocols with soft-state mechanisms (using RSVP as the running example) and to analyse the model using state spaces [14] and their associated *Strongly Connected Component (SCC)* graphs [14]. The paper also provides an introduction to RSVP and explains the standard behavioural properties of CPNs and their application to the analysis of RSVP. Our investigation is supported by a software tool called Design/CPN [17].

The paper has been organised as follows. Section 2 presents an overview of RSVP, which includes its characteristics and operation. Section 3 summarises the major steps of the methodology we use for the specification and analysis of RSVP. The methodology requires the definition of a set of RSVP Service Primitives, and these are given in Section 4. Section 5 presents the scope and assumptions of the model. Section 6 gives an informal introduction to the components, dynamic behaviour and hierarchical structuring mechanism of CPNs, illustrated by part of the CPN model of RSVP. We assume the reader is familiar with basic Petri Net notation. Section 7 describes the state space method and its application to the analysis of the model of RSVP. Section 8 introduces the standard properties of CPNs and how they can be used to demonstrate that the protocol works as expected. In Section 9, the RSVP model is analysed for a more complete set of the protocol's features. Finally, Section 10 concludes the paper.

2 Resource Reservation Protocol (RSVP) Overview

RSVP is designed to be run on network routers and in end hosts to support QoS applications. It reserves resources for a *data flow* from the sender to one or more destinations (i.e. a multicast destination). A data flow is a distinguishable packet stream, which results from using a single application (such as video conferencing) requiring a certain QoS. A packet stream includes all packets that travel from the same source to the same destination. Unlike other signalling protocols [9], RSVP destinations (receivers)

request resource reservations. Those requests travel on the reverse path of the data flow by following the pre-established route setup by RSVP [8]. RSVP is also responsible for maintaining reservations on each node (i.e. host or router) associated with the data flow. RSVP uses a soft-state approach where the reservation states must be refreshed periodically; otherwise they are automatically removed. The approach accommodates dynamic route changes, dynamic multicast group membership and dynamic QoS changes [8]. RSVP reserves resources for a *session*. A session includes all data flows from one or more senders to the same unicast (one receiver) or multicast destination (multiple receivers).

RSVP reservation requests are defined in terms of a *filter specification (filter spec)* and a *flow specification (flow spec)* [8][9]. A filter spec is used to identify the data flow that is to receive the QoS specified in a flow specification. A flow spec defines the desired QoS in terms of a service class, which comprises a *Reservation Specification (RSpec)*, and a *Traffic Specification (TSpec)*. A *RSpec* defines the reservation characteristics (i.e. the desired QoS) of the flow, for example, the service rate the application requests. A *TSpec* defines the traffic characteristics of the flow, for example, the peak data rate.

RSVP uses several messages in order to create, maintain and release state information for a session between one or more senders and one or more receivers as shown in Figure 1. We now describe the main features of RSVP, structured to facilitate the description of the model.

Figure 1: RSVP messages.

- a. **Path Setup:** In RSVP, reservation requests travel from receivers to the sender(s). Thus they flow in the opposite direction to the user data flow for which such reservations are being requested. *Path messages* are used by the sender to set up a route to be followed by the reservation requests. This path uses the same routers as the corresponding data flow. Path messages travel downstream and set up and maintain path state information (e.g. the Internet Protocol address of the previous router and the data flow's traffic characteristics).
- b. **Path Refresh:** Path and reservation states have two timers associated with them: a *refresh timer* and a *cleanup timer*. A refresh timer determines when a path or reservation refresh message will be generated. The cleanup timer determines the maximum period of time that a node (i.e. router or host) can wait to receive a path or reservation refresh message, before it removes the associated state information. A path refresh is the result of either a path refresh timeout or a user request to modify the path state. Once a path is established, a node sends path refresh messages (i.e. *Path messages*) periodically (i.e. every refresh timeout period [8]).
- c. **Path Error:** A node that detects an error in a Path message, generates and sends a *PathErr message* upstream towards the sender that created the error. It travels hop-by-hop to the sender and does not modify any path state at the nodes through which it passes. Once the sender receives the *PathErr message*, it can report the error to the application, which may take corrective action.
- d. **Path Release:** RSVP tear down messages are intended to speed up the removal of path and reservation state information from the nodes. They may be triggered because a cleanup timeout

occurs or an application wishes to finish a session. A *PathTear message* travels downstream from a sender to the receiver(s) and deletes any path state information and dependent resource reservation associated with the session and sender.

- e. **Reservation Setup:** *Resv messages* carry reservation requests (e.g. for bandwidth and buffers) used to set up reservation state information in the nodes of the route established by the path set-up message. They travel upstream from the receiver(s) to the sender(s). Reservation requests from different receivers may be *merged* on arrival at a router. The aim of merging is to control the overhead of reservation messages by making them carry more than one flow and filter specification [8]. Thus, the effective filter and flow specifications, which are carried in a reservation message, are the result of merging reservations from several requests.
- f. **Reservation Refresh:** A reservation refresh is the result of either a reservation state refresh timeout or a receiver request to modify the reservation. Like path states, reservation states need to be refreshed. Thus, a receiver periodically sends reservation refresh messages (i.e. *Resv messages*) to the sender.
- g. **Reservation Release:** *ResvTear messages* travel from the receiver(s) to the sender and remove any reservation state information associated with the receiver's data flow.
- h. **Reservation Error:** If a node detects an error in a *Resv message*, it sends a *ResvErr message* downstream to the receiver that generated the failed *Resv message*. Processing *ResvErr messages* does not result in the removal of any reservation state.
- i. **Reservation Confirmation:** Optionally, a receiver may ask for confirmation of its reservation. A *ResvConf message* is used to notify the receiver that the reservation request was successful. In the simplest case, a *ResvConf message* is generated by the sender (Figure 1).

3 Protocol Verification Methodology

Verification of RSVP firstly requires a formal specification of the protocol and the service it is intended to supply to its users [2]. This is set in the context of a protocol architecture, which in the case of RSVP, is given by the IntServ architecture [7]. A *service specification* [13] defines a set of events, known as *service primitives* (see Section 4), and their possible sequences at the interfaces between the users (an application or higher level protocol entity) and the *service provider* (comprising the protocol entities concerned and their communication mechanisms). RSVP [8] does not include an explicit service specification, but it does include an application interface from which the RSVP service specification [29][30] has been derived, taking into account the features of the protocol specification. A *Protocol Specification* formally describes the features of the protocol. RFC 2205 [8] provides the source document from which the RSVP formal model is derived. Both the service and protocol specifications are defined using Coloured Petri Nets.

Once the service and protocol specifications are defined, the protocol can be analysed for general properties and specific properties defined for RSVP. It can also be compared with the service specification, to see if the sequences of primitives defined in the service specification are preserved in the protocol specification.

4 RSVP Service Primitives

Service primitives provide an abstract way to describe the interaction between each RSVP service user (ie QoS-aware application) and RSVP service provider in an implementation independent way (see Figure 2)

[13]. A QoS-aware application must interact with RSVP in order to get some services such as a reservation request for a data flow. RSVP is the service provider. RSVP entities at each RSVP-aware router along the path of the data flow interact in order to provide the requested service.

Figure 2: RSVP as a service provider.

RSVP specification provides several generic interfaces between RSVP and other protocols and mechanisms [8]. Those interfaces are particular to a real implementation. The proposed RSVP service specification, otherwise, is intended to provide a more abstract way to describe the interaction between the application (ie service user) and RSVP (ie the service provider). Thus, several service primitives for RSVP have been defined based on the Application/RSVP interface described in [8] and the protocol specification (ie by using *service abstraction*) [8][30].

Each primitive can be either a request or an indication. A *request (Req)* is used for the application to ask for a service from RSVP. An *indication (Ind)* is used by RSVP to notify the application of the invocation of a request primitive by the other peer at the other end or that RSVP generated an event.

Table 1 shows the list of the service primitives that have been defined for RSVP and the corresponding Application/RSVP calls. They are described as follows:

- a. **RSVP-Sender (Req/Ind):** a sender application uses this primitive to establish or update the traffic characteristics of a data flow for a RSVP session.
- b. **RSVP-Reserve (Req/Ind):** a receiver application uses this primitive to establish or to modify a resource reservation during a session.
- c. **RSVP-SenderRel (Req/Ind):** a sender application uses this primitive to close a session. This means that the user data flow will eventually not have any QoS reserved.
- d. **RSVP-ReceiverRel (Req/Ind):** a receiver application uses this primitive to close a quality controlled session.
- e. **RSVP-ResvConf (Ind):** is used by the service provider to confirm that a reservation has been made.
- f. **RSVP-SenderError (Ind):** is used by the service provider to report an error in propagation or installation of the Sender's traffic characteristics to the sender.
- g. **RSVP-ResvError (Ind):** is used by the service provider to report a reservation failure inside the network to the receiver.

Generic name	Specific Name	Application/ RSVP interface name
RSVP-Sender	Request Indication	Sender PathEvent Upcall
RSVP-Reserve	Request Indication	Reserve ResvEvent Upcall
RSVP-RelSender	Request Indication	RelSender Not Defined
RSVP-RelReceiver	Request Indication	RelReserve Not Defined
RSVP-ChangeResv	Request Indication	Reserve ResvEvent Upcall
RSVP-ChangeSender	Request Indication	Sender PathEvent Upcall
Provided-Initiated Service		
RSVP-ResvConf	Indication	ResvConf Upcall
RSVP-SenderError	Indication	PathErr Upcall
RSVP-ResvError	Indication	ResvErr Upcall

Table 1: RSVP Service Primitives.

5 Modelling Scope and Assumptions

Since RSVP is a complex protocol, the scope of the model is limited to make analysis tractable. The CPN model has been developed based on the protocol specification [8] and includes most of RSVP's features. The network topology comprises two hosts (sender and receiver) and a single router between them (Figure 1). This allows us to consider router behaviour in RSVP's operation. Although multiple RSVP sessions can be open simultaneously, without loss of generality, just one session is modelled to study the functional behaviour of RSVP, since RSVP treats each session independently. Following an incremental methodology, it is important to firstly consider a unicast network. The multicast case will be considered in future work.

Although the Internet Protocol (IP) over which RSVP operates, does not guarantee that the order of sending messages is maintained at the receiver, message overtaking is only included in one of the models (see Section 9.1). This is because we want to be sure that RSVP will work under normal conditions (re-ordering is a rare event), before considering unusual events. Similarly IP may lose messages. As mentioned before, RSVP Path and Resv refresh messages deal with occasional loss of RSVP messages. Although, message loss is not considered in the model presented here, the mechanisms for dealing with loss are modelled.

6 Coloured Petri Net Modelling

Coloured Petri Nets (CPNs) [14][16] provide compact descriptions of concurrent systems by including abstract data types within the basic Petri net framework. We describe CPNs and how they can be used for modelling RSVP through an example [30]. The example is based on RSVP's path management features (i.e. path setup, refresh, error and release) for a sending protocol entity, as shown in Figure 3.

6.1 Places

In Figure 3, there are four *places* drawn as ellipses. The Sender place stores the state of the sending protocol entity. The places SOutgoingMsgs and SIncomingMsgs store RSVP messages travelling downstream and upstream, respectively. Finally, the place SenderUser represents the sending application, which requires RSVP services. Each place has an associated *type* which is written in italics at the top right of the place. Place types are defined in a set of declarations shown in Figure 4. For example, place Sender is typed by *SenderState*, defined at the top of Figure 4 using the syntax of Design/CPN, which is called CPN ML, a variant of Standard ML.

Figure 3: Path management.

A *Marking* of a place defines a collection of data values, known as *tokens*, that are associated with that place. The value is taken from the type of the place. This collection of tokens is a multi-set, since it may contain several tokens of the same value. CPNs also include the initial state of the system, called the *initial marking*. The initial marking (if not the empty multi-set) is written near each place. In the initial marking of the example, the place SenderUser contains the requested traffic characteristics, in this case, represented symbolically as 1'Ta, as the protocol mechanisms are not affected by the details of the traffic characteristics. Each communication place is empty (hence there is no inscription). The state place, Sender, contains a triple comprising null entries for path and reservation information and SESSION for the status of the sender, indicating that the sender is about to commence a RSVP session.

6.2 Transitions

Transitions represent atomic events of the system. They are drawn as rectangles in Figure 3. They may also have *guards* associated with them, which are included in square brackets. Guards are Boolean expressions which are important for describing CPN dynamics and are discussed later in this section. There are five transitions in the example. The transition RSVPSENDERReq models the establishment or updating of path state information and the occurrence of the service primitive RSVP-SenderReq. The transition PathRfrTimeOut models periodic path refreshes required to maintain path state information. The RSVPSENDERRelReq transition models the action taken when the sender user leaves the session on the occurrence of the service primitive RSVP-SenderRelReq. The transition RSVPSENDERErrorInd

represents an occurrence of the service primitive RSVP-SenderError.Ind. Finally, DiscardError removes path error messages that are not reported to the application.

```
(* ===== States of RSVP entities ===== *)

color ParameterValues      = with E|Ta|Tb|Fa|Fb;
color STSpec               = subset ParameterValues with [E,Ta,Tb];
color SFSpec               = subset ParameterValues with [E,Fa,Fb];
color Status               = with SESSION|IDLE|WAITINGRESV|RESVREADY|
                             RESVCONFIRMED|CLOSED;
color SenderStatus         = subset Status with [SESSION,WAITINGRESV,RESVREADY,CLOSED];
color SenderState          = product SenderStatus * STSpec * SFSpec;

(* ===== RSVP Messages ===== *)

color TSpec                = subset ParameterValues with [Ta,Tb];
color FSpec                = subset ParameterValues with [Fa,Fb];
color TearMsgType          = with TEARDOWN|REL;
color ResvTear             = product TearMsgType * FSpec;
color PathTear             = product TearMsgType * TSpec;
color UpstreamMessages     = union patherror: TSpec + resvtear: ResvTear + resv: FSpec;
color DownstreamMessages  = union path: TSpec + resverror: FSpec + resvconf: FSpec +
                             pathtear: PathTear;

(* ===== Variables ===== *)

var sta: Status;
var tspec,tspec1: STSpec;
var fspec,fspec1: SFSpec;

(* ===== Functions ===== *)

fun pathexists (sn) = (sn= WAITINGRESV orelse sn=RESVREADY orelse sn = RESVCONFIRMED);
```

Figure 4: Part of the global declaration of the RSVP model.

6.3 Arcs

Arcs connect transitions to places and places to transitions and are represented by arrows. A transition may have input places connected by incoming arcs and output places connected by outgoing arcs. Arcs have *expressions* associated with them. The expressions are built from constants, variables and functions and are written next to their associated arcs using CPN ML. The functions are defined, and the constants and variables declared, in the set of declarations (see Figure 4).

6.4 Declarations

Types, variables and functions are defined in what is called the *declarations* of a CPN. A part of the declarations, including the definitions that are relevant to Figure 3, are shown in Figure 4. As already mentioned, they are written in the functional programming language ML [22]. The variant used in Design/CPN, known as CPN ML, has some special key words. Here *color* is used to denote a type. The declarations are divided into 4 sections.

States of RSVP Entities. ParameterValues is an enumerated type, which represents abstract values for both the *traffic specification* (*tspec*) and *flow specification* (*fspec*) parameters. The possible values are Ta, Tb, Fa, Fb and E (empty). The types, STSpec and SFSpec, are subsets of ParameterValues and represent the traffic specification stored as part of the path state information and the flow specification stored as part of the reservation state information, respectively. Status is an enumerated type, which defines the states of the RSVP entities as follows:

- a. **SESSION:** the sender or receiver has opened a session, but no path or reservation has yet been established.
- b. **IDLE:** there exists neither path nor reservation information at the router.
- c. **WAITINGRESV:** a request with the Sender's traffic information has been accepted by the entity (i.e. sender, router or receiver) and sent (if the entity is not the receiver) but as yet no reservation request has been received.
- d. **RESVREADY:** a reservation request has been accepted and sent (if the entity is not the Sender).
- e. **RESVCONFIRMED:** a reservation has been established and a confirmation has been received.
- f. **CLOSED:** the sender or the receiver has left the session.

SenderStatus only requires four of these states and is a subset of Status. SenderState represents the states of the sender and is the product of SenderStatus, STSpec and SFSpec. The Sender place has the type SenderState, as previously mentioned.

RSVP Messages. TSpec and FSpec, represent the parameters that may be carried in RSVP messages [8] and are the traffic specification and flow specification respectively. TearMsgType is intended to distinguish between: a PathTear message generated as a result of the sender or receiver leaving the session (REL); or a path or reservation cleanup time-out (TEARDOWN). The other seven RSVP messages defined in Section 2 are represented by UpstreamMessages and DownstreamMessages. The places SOutgoingMsgs and SIncomingMsgs are typed by DownstreamMessages and UpstreamMessages, respectively.

Variables and Functions. The variables used in CPN inscriptions are typed in the declarations. As an example, the variable *sta* represents the status of the RSVP entity (e.g. the sender). The function *pathexists* is used to simplify guard inscriptions and returns true if the sender has path state information available.

6.5 Enabling and Occurrence of Transitions

Transitions can be *enabled* and can then *occur*. A transition is enabled if its input places have the required tokens and its *guard* is true. These enabling requirements are determined by *binding* the transition's variables to values taken from their types. The choice of binding value is arbitrary. The required tokens are defined by evaluating the input arc expressions for a particular binding of the variables. This same binding is used for evaluating the guard. The occurrence of a transition removes tokens from its input places and adds tokens to its output places. The removed tokens are defined by the evaluated expressions on the corresponding incoming arcs for this binding of variables, while the values of the added tokens are determined by evaluating the arc expressions on the corresponding outgoing arcs for the same binding. Hence transitions can occur in different modes, depending on the bindings of the variables.

For example, in Figure 3, RSVP-Sender.Req is enabled when the Sender is not CLOSED (see the guard), and a new *tspec* is waiting to be sent in SenderUser. An occurrence of the transition RSVP-Sender.Req updates the *tspec* to the one requested by the user (ie to Ta). If the status of the sender is equal to SESSION, it is also updated so that it indicates that the sender is ready to receive a reservation request (WAITINGRESV). If not, the state of the sender is only updated by the new *tspec* value Ta. In addition, Ta is removed from the place SenderUser and a Path message carrying the corresponding *tspec* = Ta, is added to the SOutgoingMsgs place.

6.6 Hierarchical CPNs

We deal with RSVP's complexity by using the hierarchical constructs of CPNs [14][16]. Hierarchies are built using the notion of a *substitution transition*, which may be considered a macro expansion. The

model starts with a top-level CPN diagram, which provides an overview of the system being modelled and its environment. In hierarchical CPNs, this top-level diagram will contain a number of substitution transitions. Each of these substitution transitions is then refined by another CPN diagram, which may also contain substitution transitions. The top-level diagram and each of the substitution transitions is defined by a module, called a *page*. The relationships between the different pages are defined by a *hierarchy page*. The hierarchy page also includes the name of the page that defines the declarations required for the CPN inscriptions, called the *Global Declaration node* (see Section 6.4).

Figure 5: Hierarchy page of the RSVP CPN model.

The hierarchy page of the RSVP CPN model consists of eleven (11) pages as illustrated in Figure 5. Page 11 represents the Global Declaration node (Figure 4). The top-level page is called RSVPNetwork and is shown in Figure 6. It shows the network topology including interaction with the applications. The three shaded rectangles (Sender, Router and Receiver) represent RSVP entities and are substitution transitions. A substitution transition is identified by a HS-tag in the lower left corner of the rectangle representing the transition. The ellipse, SenderUser, is described in Section 6.1. Similarly the place ReceiverUser is typed by FSpec and contains flow specifications requested by the receiver user. The place RIncomingMsgs has markings that represent RSVP messages travelling downstream, while ROutgoingMsgs has markings that represent RSVP messages travelling upstream. The other places are described in Section 6.1.

The substitution transitions *Sender*, *Router* and *Receiver* are defined by their own pages and comprise substitution transitions for Path and Reservation management, which are defined at the lowest level of the hierarchy. These correspond with the major functions of RSVP described in Section 2. The Path and Resv management pages include transitions that model the establishment, refreshment, release and error control of paths and reservations, respectively. The transitions model service primitives and protocol actions implementing RSVP functions (eg path refresh) or discarding messages that cannot be processed. For example, the Sender page (Figure 7) includes two substitution transitions, PathManagement (see Figure 3) and ResvManagement (see Figure 8) and a new place Sender. The place Sender is typed by SenderState

and models the status of the sender together with path and reservation information. The other places were described in Figure 6.

Figure 6: RSVP Network page.

Figure 7: Sender page.

The (Sender) ResvManagement page is shown in Figure 8. The place NoRcvUser is typed by UserInd and indicates whether the sender user has left the session (NOUSR) or not (USR). The place ResvInd is typed by FspecXFlag and indicates whether the RSVP-ReserveInd service primitive has occurred for a reservation request (i.e. (fspec,1)) or not (i.e. (fspec,0)). They are used to control the occurrence of RSVP service primitives (see Section 4). The transitions RSVPReserveInd and Reserve model the establishment or updating of a reservation. RSVPReserveInd also models the occurrence of the service primitive RSVP-ReserveInd and the confirmation of the reservation.

Figure 8: Resv Management page.

Transitions ResvSetupFailed and NoPathInfo model a failed reservation setup due to insufficient network resources and no path information available, respectively. They return a ResvErr message to the receiver. RefreshResv models a reservation refresh. RSVPreceiverRelInd and ResvTear model the actions taken when a ResvTear message is received. RSVPreceiverRelInd also models the occurrence of the service primitive RSVP-ReceiverRel.Ind. ResvCleanup models the deletion of the reservation at the sender due to the expiration of the resv cleanup timer. Finally, Discard drops an incoming ResvTear message when the Sender is not ready to process it.

7 Analysis of CPNs

7.1 Modifying the CPN Model for Analysis

To analyse the CPN model we employ *state space* methods [14]. An *Occurrence Graph (OG)* can be generated using Design/CPN [25]. It includes all possible markings that can be reached from the initial marking and is represented by a directed graph where the nodes represent the markings and the edges the occurring *binding elements* comprising the CPN transition and the assignment to the transition's variables.

The CPN model of RSVP can generate an infinite state space when path and reservation refreshes are included (e.g. see transition PathRfrTimeOut in Figure 3), since an arbitrary number of RSVP messages can be added to the communication places (e.g. SOutgoingMsgs). To gain insight into the behaviour of the CPN model we would like to explore a finite state space. This can be achieved by limiting the number of refreshes that can occur (which we consider to be a severe limitation on the operation of RSVP) or by limiting the storage capacity of the communication network. Given that storage in the network is finite,

this is a more realistic option. We thus modify the model so that the communication places have finite capacity.

The types, DownstreamMessages and UpstreamMessages, are modified to include a *no message* value. Figure 9 illustrates how the Sender's Path Management page (see Figure 3) has been modified to limit the capacities of the communication places using the notion of empty message slots. The Sender can send a message into the network, (e.g. see transition *RSVPSenderReq*) if *SOutgoingMsgs* contains a token indicating the presence of an empty slot (i.e. $1'nodmsg(N)$). Also, when the sender receives a message (e.g. see transition *RSVPSenderErrorInd*) it adds a token representing an empty slot (i.e. $1'nomsg(N)$) to *SIncomingMsgs*. We also modify the model to ensure that the sequence of requests for traffic specification changes by the sender user is maintained, by using a list type. This reduces the state space, by not having to consider arbitrary choices of symbolic traffic characteristics, which are of no importance from an analysis perspective. In addition, we have split transition *RSVPSenderRelReq* (of Figure 3) into two transitions (*RSVPSenderRelReq* and *RSVPSenderRelReq2*) so that its occurrence does not depend on the availability of a token on the *SOutgoingMsgs* place when the Sender has not yet sent a path message. In this case it can leave the session without being concerned about the state of the network. If the transition was not split, this occurrence of *RSVPSenderRelReq* would depend on their being an empty slot in the network, which clearly is not relevant to the sender, as no Path Tear message needs to be sent to the network in this case.

Figure 9: Modified Path Management page.

7.2 State Space Analysis

Figure 10 shows the state space for our model of RSVP when similar changes are also incorporated into the Router and Receiver parts of the model. To obtain a small state space suitable for presentation we have restricted the model to only include the basic features of path and reservation establishment. Our aim here is to provide an illustration of the approach, rather than the full analysis, for which the reader is referred to [30].

Figure 10: State space of the example.

There are nineteen (19) reachable markings (or nodes), represented by rounded boxes in the figure. Each marking has an identification number located at the top. Also, there are two numbers separated by a colon ("."), which represent the number of input and output arcs, respectively. The node at the top of the figure is the initial marking and has the identification number 1.

The details of each of the markings can be obtained easily from Design/CPN and are shown in the dashed boxes next to the nodes for the initial marking and for each of the terminal markings (leaf nodes). For example, in the initial marking (i.e. node 1): the Router is idle (IDLE,E,E) and the Sender and Receiver are in the same session (SESSION,E,E); each of the communication places (SIncomingMsgs, ROutgoingMsgs, SOutgoingMsgs and RIncomingMsgs) has an empty slot (i.e. 1'nodmsgs (N) or 1'nodmsgs (N)); and the SenderUser is ready to provide its traffic characteristics (1'[Ta]) while the ReceiverUser desires that a reservation (1'[Fa]) be made.

An arc represents the occurrence of a *binding element*. The details of binding elements can be shown on the arcs, which we have done for several of them. Also included is the identification number of the arc (located in the upper left corner) and the related node numbers ($n1 \rightarrow n2$, indicates that the marking $n2$ is reached from marking $n1$ when the binding element occurs). For example, the binding element 1

(represented by arc 1) is the only one enabled in the initial marking. It includes the transition RSVPSENDERReq and the bindings: tspeclist = [] (no more sender requests), tspec1 = E (no traffic information is initially installed at the sender), tspec = Ta (the sender informs RSVP of the required traffic characteristics equal to Ta), sta = SESSION (the status of the sender is SESSION), fspec = E (no reservation information is installed at the sender).

In Figure 10, the path defined by the node sequence 1,2,3,6,9,11,14,17,19 shows the sequences of events leading to the successful establishment of a reservation, while the path 1,2,3,6,11,13,16,18 illustrates an event sequence where the reservation is not set up in the sender. The Sender couldn't establish the reservation (see arc 12) because, for example, it didn't have enough resources to satisfy the request. The other leaf nodes of the graph (7, 10 and 15) represent situations where the path is not established (7,10), or the reservation fails (15). The terminal nodes show that the completion of RSVP execution satisfies the following expected conditions: RSVP finishes in a state where all the message buffers are empty (represented by the tokens noumsgs (N) and nodmsgs (N)) and a path state has not been installed in at least one of the RSVP entities because the sender request has failed in one RSVP node, so no reservation exists (i.e. the RSVP entity, e.g. the Router, is not in the WAITINGRESV state) (e.g. see node 7). Alternatively, a reservation state has not been established in at least one of the RSVP entities (i.e. it is not in the RESVREADY or RESVCONFIRMED state) because the reservation request has failed in one RSVP node (e.g. see node 15); or a reservation state has been set up in all RSVP entities along the route of the data flow (i.e. they are in the RESVREADY or RESVCONFIRMED state) (see node 19).

7.3 Strongly Connected Components

A *Strongly Connected Component (SCC)* of the occurrence graph is a maximal sub-graph, whose nodes are mutually reachable from each other [14]. A SCC graph has a node for each SCC and arcs that connect each SCC with other SCCs. A SCC without incoming arcs is called the *initial SCC*, and a SCC without outgoing arcs is called a *terminal SCC*. Each node in the state space belongs to only one SCC, so the SCC graph will never have more nodes than the corresponding occurrence graph.

The SCC graph of the OG of Figure 10 will be the same as Figure 10, as it contains no cycles. To illustrate a non-trivial SCC graph we generate an OG for a RSVP model in which path refreshes are generated at the sender but no reservation request is sent by the receiver. The full state space has 20 nodes and 21 arcs, while the SCC graph has 6 nodes and 13 arcs as shown in Figure 11. Node number ~1 is the initial SCC. Terminal SCC nodes are indicated by solid boxes (i.e. nodes ~4 and ~6). Each SCC node has information about the number of OG nodes and arcs that comprise the associated SCC. The number of OG arcs from one SCC node to another, is indicated next to the SCC graph arc. For example, node ~2 comprises four markings (OG nodes) and 5 OG arcs, one input arc and two output arcs to SCC node ~3.

In the RSVP model the cycles indicated by the SCC graph are the result of refreshes. Terminal SCC nodes represent protocol sequences that can be executed indefinitely, possibly without making effective progress (in which case livelocks exist). Thus they must be checked to see if these sequences are expected. In this case, these sequences must correspond to either successful path establishment or to a path establishment attempt that failed at the receiver. The terminal SCC sequences can be easily checked using standard Design/CPN functions [17] that are provided to explore OG nodes included in a SCC node. We have checked them (see [30]) and they are as expected.

Figure 11: Example illustrating a SCC graph.

8 Behavioural Properties of Coloured Petri Nets

This section introduces in an informal way the main behaviour or dynamic properties of CPNs and how they can be used for the analysis of RSVP. They describe the expected behaviour of the model. More details about these properties and their formal definitions can be found in [14].

8.1 Reachability

By convention, M_n denotes the marking of node number n . M_n is reachable from M_1 if there is an occurrence sequence from marking M_1 to M_n . We have defined a set of desired properties that describe the expected behaviour of RSVP [30]. These are so called *safety properties*, which we have checked by establishing that the desired markings are reachable. The properties cover the setting up, maintenance and release of both paths and reservations. We have recently published [31] how these properties can be formalised and proved using ML queries on the OG. Here, we illustrate how the state space can be used to analyse RSVP for these specific properties by considering two of these properties: path set-up and reservation set-up.

Once the RSVP-Sender.Req service primitive occurs, RSVP should be able to establish or update the path state in all RSVP nodes on the route of the data flow, unless the receiver has closed the session before the RSVP-Sender.Req service primitive occurs. For example, in Figure 10, the destination marking of the occurrence of the transition `RSVPSENDERReq` (marking 2) can reach marking 6 shown in Table 2. It indicates that the Sender, Router and Receiver are in the WAITINGRESV state and the traffic characteristics, which have been installed, are T_a .

Place	Marking
Router'Router 1	1'(WAITINGRESV,Ta,E)
Receiver'Receiver 1	1'(WAITINGRESV,Ta,E)
Sender'Sender 1	1'(WAITINGRESV,Ta,E)
RSVPNetwork'SOutgoingMsgs 1	1'nodmsg(N)
RSVPNetwork'SIncomingMsgs 1	1'noumsg(N)
RSVPNetwork'RIIncomingMsgs 1	1'nodmsg(N)
RSVPNetwork'ROutgoingMsgs 1	1'noumsg(N)
RSVPNetwork'SenderUser 1	1'[]
RSVPNetwork'ReceiverUser 1	1'[Fa]

Table 2: Marking 6 of Figure 10.

Another important property of RSVP is *Resv Setup*, which is defined as follows. If the RSVP-ReserveReq service primitive occurs, RSVP should be able to establish or update a reservation along the route of the data flow, unless the sender has left the session before the reservation request is sent. Also, the corresponding path state information must exist at all RSVP nodes along the route of the data flow before the reservation is established. In Figure 10, we see that the reservation request primitive occurs in marking 6, and that the property is satisfied by marking 19. It indicates that the Sender and Router are in RESVREADY, the Receiver is in RESVCONFIRMED and the reservation which has been installed is Fa. We see that marking 19 is reachable from marking 9 (the resultant marking on the occurrence of the Reservation request in marking 6) by the following sequence of markings: 9,11,14,17,19.

As the size of the state space increases, it is not possible to verify these safety properties using visual inspection. We developed the algorithm *Reachable* [30] to check if multiple nodes can reach at least one of the nodes in a list, since the in-built function provided by Design/CPN [17] only checks the reachability of one node from another. The algorithm is used to check RSVP's safety properties such as the Resv Setup property just explained.

8.2 Boundedness

The *upper* and *lower integer bounds* indicate the maximum and minimum number of tokens that can be located on each place in the reachable markings. For example, they are useful for detecting unbounded communication buffers. In Figure 10, the upper and lower bounds for the communication places are 1. This means that each of these places can either contain one message or an empty slot (represented by the token nodmsg (N) or noumsg (N)).

The other concept related to boundedness properties is *multi-set bounds*. They provide information about the value of the tokens that the places can carry. The *upper multi-set bound* of a place is defined as the smallest multi-set which is larger than or equal to all reachable markings of the place [14]. The *lower multi-set bound* of a place is defined as the largest multi-set which is smaller than or equal to all reachable markings of the place. For example, the best upper multi-set bound of the place Sender is: $1'(SESSION,E,E)++1'(WAITINGRESV,Ta,E)++1'(RESVREADY,Ta,Fa)$. This indicates that the Sender can be in all the defined states (see Section 6.4) except for CLOSED because the sender release feature (i.e. the transition RSVPSEnderRel in Figure 3) has been switched off.

8.3 Home Markings

A *home marking* is a marking that can always be reached from all other reachable markings. In Figure 10, it can be seen that there is no home marking. A *home space* is a set of markings such that from each reachable marking, it is possible to reach at least one of these markings. Although there is no home

marking in this model, the markings M7,M10,M15,M18 and M19 form a home space. This means that the protocol can always finish in an expected state (see Section 7.2).

8.4 Deadlock-Freeness

A dead marking is a marking with no enabled binding elements. In Figure 10, M7,M10,M15,M18 and M19 are dead markings, however they are expected (see Section 7.2) and hence are not deadlocks from an RSVP perspective.

8.5 Dead Transitions

A *dead transition* is not enabled in any reachable marking. When generating the state space shown in Figure 10, we have deactivated several transitions to make the state space small. For example, the refresh transitions, such as PathRfrTimeOut (see Figure 9), do not occur in the example, so they are dead transitions. Dead transitions represent protocol actions that never happen. This situation is not acceptable, so dead transitions must be analysed to see if they are the result of a modelling problem or a protocol specification problem.

9 Further Analysis of the RSVP Model

In the course of our investigations [30], a series of state spaces was generated for different models, that were incrementally developed to include more features, and for different initial markings. This was to gain further confidence in the model as it was developed. In this section, we introduce these models, however, we only analyse the RSVP model, which includes the complete set of RSVP features considered in this work. It is verified against the behavioural properties of CPNs presented in Section 8 following the analysis approach described in Section 7. As mentioned before, Design/CPN [17] is used to simulate and analyse the model. Design/CPN was run on a 1.6 GHz Linux Pentium 4 PC with 1 GB RAM.

9.1 Classes of RSVP Models

For RSVP verification, the RSVP models were categorised as follows [30]:

1. **RSVP without Changes (RWTOC):** is the most basic model where no changes of reservations are considered. The sender user sends a request to the receiver indicating the traffic characteristics of the data. Once the receiver receives the request, it sends a reservation request to the sender.
2. **RSVP with Overtaking (RWTHO):** is the same as 1. but allows RSVP messages to arrive at their destinations out of order (i.e. messages can be overtaken).
3. **RSVP with Changes (RWTHC):** considers dynamic changes of traffic and reservation information by the users but not message overtaking.

In addition, the various features of RSVP (see Section 2) were added to these models based on how complex the model analysis becomes when features were added as shown in Table 3 and Table 4 . Note that the suffix “C” indicates that the model belongs to the RSVP with Changes model.

Features	Model Classes							
	RSVP without Changes				RSVP with Overtaking			
	RS	RR	RC	RRI	RS	RR	RC	RRI
Path Setup	✓	✓	✓	✓	✓	✓	✓	✓
Path Changes								
Path Refresh		✓	✓	✓		✓	✓	✓
Path Cleanup			✓	✓			✓	✓
Path Error	✓	✓	✓	✓	✓	✓	✓	✓
Sender Release				✓				✓
Resv Setup	✓	✓	✓	✓	✓	✓	✓	✓
Resv Changes								
Resv Refresh		✓	✓	✓		✓	✓	✓
Resv Cleanup			✓	✓			✓	✓
Resv Error	✓	✓	✓	✓	✓	✓	✓	✓
Receiver Release				✓				✓

RS: Resv Setup; RR: Resv Refresh; RC: Resv Cleanup; RRI: Resv Release

Table 3: RSVP models - part I.

Features	Model classes			
	RSVP with Changes			
	RS-C	RR-C	RC-C	RRI-C
Path Setup	✓	✓	✓	✓
Path Changes	✓	✓	✓	✓
Path Refresh		✓	✓	✓
Path Cleanup			✓	✓
Path Error	✓	✓	✓	✓
Sender Release				✓
Resv Setup	✓	✓	✓	✓
Resv Changes	✓	✓	✓	✓
Resv Refresh		✓	✓	✓
Resv Cleanup			✓	✓
Resv Error	✓	✓	✓	✓
Receiver Release				✓

RS-C: Resv Setup (change); RR-C: Resv Refresh (change); RC-C: Resv Cleanup (change); RRI-C: Resv Release (change)

Table 4: RSVP models - part II.

9.2 Initialisation

Each RSVP model is initialised by distributing tokens to one or more places of the model to create the initial marking. The initial marking of each model is given in [30]. In this section, we only present the initialisation of the RSVP model, which we analyse in this section (i.e. the RSVP with Changes (RRI-C) model). Each of the places SenderUser and ReceiverUser contains a list of the requested traffic or reservation characteristics, $1'[Ta, Tb]$ and $1'[Fa, Fb]$, respectively. Each communication place contains a single *no message* token (*nodmsg* or *noumsg*), indicating it is empty. Each state place (Sender, Router and Receiver) contains a triple comprising null entries (ie E) for path and reservation information and SESSION for the status of the sender and receiver and IDLE for the router status.

9.3 Occurrence Graph and SCC Graph Statistics

The occurrence graphs are generated for the models described in Section 9.1. The number of nodes and arcs of the full OGs and SCC graphs and the corresponding generation times are shown in Table 5 and Table 6.

Statistical Information/Models	RS		RR		RC		RRI	
	RWTOC	RWTHO	RWTOC	RWTHO	RWTOC	RWTHO	RWTOC	RWTHO
OG								
Number of Nodes	19	19	281	1095	3236	14610	6459	42848
Number of Arcs	18	18	1049	5225	15488	83720	31280	235723
Generation time (mm:ss)	00	00	01	05	24	2:55	55	24:07
SCC graph								
Number of Nodes	19	19	14	19	298	626	2025	12903
Number of Arcs	18	18	80	330	2481	10086	13917	107675
Generation Time (mm:ss)	00	00	00	00	00	07	04	01:20

Table 5: Occurrence graph statistics of the RSVP without changes and RSVP with Overtaking models.

Statistical Information / Models	RWTHC model			
	RS-C	RR-C	RC-C	RRI-C
OG				
Number of Nodes	499	6292	77751	154986
Number of Arcs	866	23731	376103	769880
Generation Time (hh:mm:ss)	00:00:01	00:00:46	00:46:14	3:27:20
SCC graph				
Number of Nodes	499	598	15066	56962
Number of Arcs	866	4725	136625	428644
Generation Time (mm:ss)	00:00	00:02	01:00	05:57

Table 6: Occurrence graph statistics of the RSVP with changes models.

The size of the state space is increased when we include more features of RSVP. For example, when the Path and Resv refreshes are activated in the model, the state space has 281 nodes and 1049 arcs (see RR RWTOC model on Table 5). This is because both Path and Resv messages can be in the network simultaneously. This is not possible in the current model with the given initialisation because the Receiver must wait until it receives the first Path message to generate and send a reservation request, and only one Resv and one Path can be generated and sent to the network.

The RS/RS-C models (see Table 5 and Table 6) are the only ones that do not have cycles, since the sizes of the OGs are the same as the sizes of the corresponding SCC graphs. Each of the other models generates a SCC graph whose size is smaller than the size of the OG, so they have cycles. The cycles are due to the path and reservation refresh messages sent to avoid the removal of the existing state information. The size of the occurrence graph increases considerably when the cleanup and release features are included in the model because new messages travel upstream and downstream (e.g. PathTear messages).

It may be observed in Table 5 that the sizes of the OGs of the RS models are exactly the same, since message overtaking does not occur in the RS RWTHO model. This is because the only messages that can travel downstream and upstream are the Path and ResvErr messages and the Resv and PathErr messages respectively. The sizes of OGs of the RR, RC and RRI RWTHO models increase considerably when compared with the sizes of the corresponding RWTOC models. This is expected given the number of states, which arises from having more than one message in the communication places. In addition, the messages may be processed by the RSVP entities in an arbitrary order.

The sizes of the OGs shown on Table 6 increase considerably when compared with the RWTOC and RWTHO models. This is because new entity states and messages are generated as a result of adding another traffic request and another reservation request.

As mentioned before, in the following section, we analyse the RRI-C model which includes all of the RSVP features described in Section 2. The CPN model of Section 6 is modified as explained in Section 7.1 to make the model tractable for analysis.

9.4 General Properties

Boundedness and *liveness* were investigated to validate and debug the model and to provide insight into RSVP's behaviour. This information is included in the standard report for the state space generated by Design/CPN [17].

9.4.1 Boundedness

Integer and multi-set bounds were analysed for the places of the model. For example, Table 7 lists the upper integer and multi-set bounds for the communication places.

Each of the communication places SOutgoingMsgs, SIncomingMsgs, ROutgoingMsgs and RIncomingMsgs may have a maximum number of one token. This corresponds with the maximum capacity of the message buffer (ie one). The multi-set bounds show that all RSVP messages can be generated by the protocol. For example, the place RIncomingMsgs can contain a token representing a Path message, which carries the traffic characteristics of the data flow (Ta). It may also contain a token representing a ResvErr message, which indicates an error. The message carries the reservation characteristics (Fa) for which the request failed. Similar analysis can be derived from the multi-set bounds for the other places. As mentioned before, the tokens noumsg and nodmsg are used to control the size of the queue [16].

The boundedness results are as expected and further confirm that the model is valid.

Place	Integer Bounds	Multi-set Bounds
SOutgoingMsgs	1	1`path(Ta)++ 1`path(Tb)++ 1`resverror(Fa)++ 1`resverror(Fb)++ 1`resvconf(Fa)++ 1`resvconf(Fb)++ 1`pathtear((REL,Ta))++ 1`pathtear((REL,Tb))++ 1`nodmsg(N)
RIncomingMsgs	1	1`path(Ta)++ 1`path(Tb)++ 1`resverror(Fa)++ 1`resverror(Fb)++ 1`resvconf(Fa)++ 1`resvconf(Fb)++ 1`pathtear((TEARDOWN,Ta))++ 1`pathtear((TEARDOWN,Tb))++ 1`pathtear((REL,Ta))++ 1`pathtear((REL,Tb))++ 1`nodmsg(N)
SIncomingMsgs	1	1`patherror(Ta)++ 1`patherror(Tb)++ 1`resvtear((TEARDOWN,Fa))++ 1`resvtear((TEARDOWN,Fb))++ 1`resvtear((REL,Fa))++ 1`resvtear((REL,Fb))++ 1`resv(Fa)++ 1`resv(Fb)++ 1`noumsg(N)
ROutgoingMsgs	1	1`patherror(Ta)++ 1`patherror(Tb)++ 1`resvtear((REL,Fa))++ 1`resvtear((REL,Fb))++ 1`resv(Fa)++ 1`resv(Fb)++ 1`noumsg(N)

Table 7: Upper bounds for communication places.

9.4.2 Dead Transitions

The Design/CPN standard report showed that there are no dead transitions. This is expected as there should be no ‘dead code’ in the specification.

9.4.3 Deadlock-Freeness (Termination Property)

RSVP must finish in a state where: all the message queues are empty; any path or reservation state information has been removed from the RSVP entities (ie sender, receiver and router); and, the sender and receiver have left the session (ie are in the CLOSED state).

Table 8 includes the number of dead markings and terminal SCC nodes (ie a SCC without outgoing arcs) of the RSVP model. Design/CPN output shows that each terminal SCC node contains one marking, corresponding to a dead marking. The size of the SCC graph is smaller than the OG indicating that there are cycles. This is expected due to the soft-state mechanisms.

Statistical Information	OG graph	SCC graph
Number of Dead Markings	85	-
Number of Terminal SCC nodes	-	85

Table 8: Number of Dead Markings and Terminal SCC Nodes of the Fully Featured RSVP Model.

A large number of dead markings (85) occurs because of the possible markings of places other than those concerned with communication and RSVP entity states, such as user places (eg SenderUser) and some control places that exist in the model.

The termination property is investigated by checking the dead markings of the CPN model. It is checked by querying the state space using powerful ML [22] functions suited to the Design/CPN environment. The following ML function, *TerminalMarkings* (see below), checks that all dead markings are expected as described above and returns any unexpected terminal markings. *PredNodes* (line 1) is a pre-declared query function [25], which traverses the nodes of an OG and returns all the nodes in the search area of the state space that satisfy a specified predicate. In this case, the search area is the list of all dead markings, and the predicate function (fn n =>) is explained in the rest of this paragraph. The function

Expected_Rel_EndMarkings (lines 3,4) defines the expected markings for the RSVP entities. It uses the markings of the Sender, Router and Receiver places (obtained by the in-built function *Mark*, and converted from multisets to triples) as parameters, checks if the states are expected (lines 6 to 8), and stores the boolean result in *em*. In lines 11 to 14, it checks that all buffers are empty (as indicated by the token values *nodmsg N* and *noumsg N*) and *ands* the result with *em* (line 16). The predicate function will return *false* (line 17) if the dead marking does satisfy the expected conditions and *true* otherwise (line 18). Finally, *NoLimit* (line 19) specifies that the search continues until all dead markings have been visited.

```

1  fun TerminalMarkings() = PredNodes(ListDeadMarkings(), fn n =>
2    let
3      fun Expected_Rel_EndMarkings ((sta1,td1,fd1),(sta2,td2,fd2),(sta3,td3,fd3))=
4        sta1 = CLOSED andalso sta2 = IDLE andalso sta3 = CLOSED;
5      (* Checks markings of state places *)
6      val em = Expected_Rel_EndMarkings(ms_to_cs(Mark.Sender'Sender 1 n),
7        ms_to_cs(Mark.Router'Router 1 n),
8        ms_to_cs(Mark.Receiver'Receiver 1 n));
9    in
10     (* Returns the result of checking the communication places *)
11     if Mark.Router'RIncomingMsgs 1 n = 1`nodmsg N andalso
12       Mark.Router'ROutgoingMsgs 1 n = 1`noumsg N andalso
13       Mark.Router'SOutgoingMsgs 1 n = 1`nodmsg N andalso
14       Mark.Router'SIncomingMsgs 1 n = 1`noumsg N andalso
15     (* Returns the result of checking the markings of state places *)
16     em
17     then false
18     else true
19     end,NoLimit);

```

The expected dead markings must also form a home space. Determination of a home space is provided as a standard query by Design/CPN [25]. The results obtained after running the *TerminalMarkings* function indicate that all dead markings are expected. Also, all dead markings form a home space, so there are no *livelocks*. Livelocks [12] are protocol sequences that can be executed indefinitely without making effective progress. They are represented in the state space as cycles from which there is no escape, and can be detected as terminal SCCs, which are not dead markings.

10 Conclusions

In this paper, we have illustrated how RSVP can be formalised and analysed using Coloured Petri Nets and their computer support package called Design/CPN. It summarises part of our effort for formally specifying and analysing RSVP [30]. We have described the protocol in a graphical way using the features of CPNs. The hierarchical structuring mechanism of CPNs allows us to identify the major entities involved in the architecture (i.e. a sender, receiver and router), to specify the major modules that form the protocol (e.g. path and reservation management) and to describe the network topology considered in the example (i.e. one sender and one receiver connected by a router). The state space method of CPNs can be used to validate the model and then to verify it against a set of generic behavioural properties (such as liveness and boundedness) and properties specific to RSVP, such path and reservation set-up. The facilities provided by CPNs (and Design/CPN) allow us to create a model that provides a clear, unambiguous and precise definition of RSVP, and to check the protocol for correctness.

The main limitation found during the analysis of RSVP is that of state space explosion [26]. The problem with very large state spaces is that they cannot be generated with limited computer memory and if they can, they may be difficult to analyse. To make the analysis feasible we have limited the scope of the

model. For example, we have only considered a very simple network topology, ideal and limited communication capability between RSVP nodes, and excluded some of RSVP's functionality such as merging.

RSVP specifies time periods for generating refreshes and cleanups. These periods could be incorporated into the CPN model by allowing tokens to carry time values. Each time value is called a time stamp. Timed CPN models and simulation can be used to analyse the performance of the system, e.g., to investigate the throughput provided by RSVP (network).

We have managed to analyse RSVP under a set of simplifying assumptions [30]. The analysis results have shown that the protocol works as expected under these assumptions. Future work in this area will attempt to relax some of these assumptions and may include modelling other features of RSVP (eg merging), more complex network topologies and inclusion of network imperfections such as message loss, since RSVP operates over the Internet Protocol which can lose, reorder or even duplicate messages. Time periods can be also incorporated to the RSVP CPN model to check if the protocol behaves as expected (i.e. functional verification) under time restrictions.

References

- [1] Babich F. and Deotto L., *Formal Methods for the Specification and Analysis of Communication Protocols*, IEEE Computing Surveys, Vol. 4, No.1, pages 2-20, third quarter 2002.
- [2] Billington J. *Formal Specification of Protocols: Protocol Engineering*. Encyclopaedia of Microcomputers, Marcel Dekker, New York, 1991, Vol. 7, pp 299-314.
- [3] Billington J., Diaz M. and Rozenberg G. (eds), *Application of Petri nets to Communication Networks*, Advances in Petri Nets, Lecture Notes in Computer Science, Vol. 1605, Springer-Verlag, 1999.
- [4] Billington J. and Han B., *Formalising the TCP Symmetrical Connection Management Service*, Proc. Design, Analysis and Simulation of Distributed Systems (DASD'03), Florida, 30 March - 3 April 2003, pp. 178-184, SCS.
- [5] Billington J. and Han B., *On Defining the Service Provided by TCP*, Proc. 26th Australian Computer Science Conference (ACSC'2003), 4-7 February, 2003, Adelaide, Australia, Conferences in Research and Practice in Information Technology, Vol.16, Australian Computer Society Inc., Adelaide, February, 2003, pp 129-138.
- [6] Billington J. and Han B., *Closed Form Expressions for the State Space of TCP's Data Transfer Service Operating over Unbounded Channels*, Proc. 27th Australasian Computer Science Conference (ACSC'2004), 18-22 January, 2004, The University of Otago, Dunedin, New Zealand, Conferences in Research and Practice in Information Technology, Vol.26, Australian Computer Society Inc., January, 2004, pp 31-39.
- [7] Braden R., Clark D., and Shenker S. *Integrated Services in the Internet Architecture: an Overview*. RFC 1633, IETF, June, 1994.
- [8] Braden R., et al. *Resource Reservation Protocol (RSVP) -- Version 1: Functional Specification*. RFC 2205, IETF, September, 1997.
- [9] Durham D. and Yavatkar R. *Inside the Internet's Resource Reservation Protocol*. Wiley, USA, 1999.
- [10] Han B. and Billington J., *An Analysis of TCP Connection Management using Coloured Petri Nets*, Proc. 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI'2001), Volume XII, pp. 590-595, Orlando, Florida, USA, 22-25 July 2001.
- [11] Han B. and Billington J., *Validating TCP Connection Management*, in Conferences in Research and Practice in Information Technology, Vol.12, Workshop on Software Engineering and Formal Methods, Australian Computer Society, Adelaide, 24-25 June 2002, pp.47-55.
- [12] Holzmann G. *Design and Validation of Computer Protocols*. Prentice Hall. 1991.

- [13] ITU-T *Convention for the Definition of OSI Services*. Recommendation X.210. 1994.
- [14] Jensen K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Vol. 1, 2 and 3. Springer-Verlag, 2nd edition, April, 1997.
- [15] Figueiredo J. and Kristensen L.M. *Using Coloured Petri Nets to Investigate Behavioural Performance Issues of TCP Protocols*. Proceedings of Second Workshop on Practical use of Coloured Petri Nets, Aarhus, Denmark, 1999.
- [16] Kristensen L.M., Christensen S., and Jensen K. *The practitioner's guide to coloured Petri nets*. International Journal on Software Tools for Technology Transfer, Springer, 1998, Vol. 2, Number 2, pp 98-132.
- [17] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*, Version 2, Meta Software Corporation, Cambridge, 1993.
- [18] Ouyang C., Kristensen, L. M. and Billington J., *A Formal Service specification for the Internet Open Trading Protocol*, Lecture Notes in Computer Science, Vol. 2360, pp.352-373, Springer Verlag, 2002.
- [19] Ouyang C., Kristensen LM and Billington J. *A Formal and Executable Specification of Internet Open Trading Protocol*. Third International Conference, LNCS 2455, pp 377-387, Springer-Verlag, 2002.
- [20] Ouyang C. and Billington J., *On verifying the Internet Open Trading Protocol*, Lecture Notes in Computer Science, Vol. 2738, pp. 292-302, Springer, 2003.
- [21] Ouyang C. and Billington J., *An Improved Formal Specification of the Internet Open Trading Protocol*, Proc. 19th Annual ACM Symposium on Applied Computing, Nicosia, Cyprus, 14-17 March 2004, pp. 779-783.
- [22] Paulson L. *ML for the Working Programmer*. Cambridge University Press. 1991.
- [23] Proceedings of the (Joint) International Conferences on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE) and Protocol Specification, Testing & Verification (PSTV), 1997-2001.
- [24] Smith M. and Ramakrishnan K. K. *Formal Verification of Safety and Performance Properties of TCP Selective Acknowledgment*. In Proceedings of 1998 International Conference on Network Protocols, Austin, Texas, October 1998.
- [25] University of Aarhus, Computer Science Department. *Design/CPN Occurrence Graph Manual*. Version 3.0. 1996.
- [26] Valmari A. *The State Explosion Problem*. Lecture Notes in Computer Science, Vol. 1491, pp 429-528, Springer-Verlag, 1998.
- [27] Vanit-Anunchai S. and Billington J., *Initial Result of a Formal Analysis of DCCP Connection Management*, accepted (16 February) for the fourth International Network Conference, Plymouth, UK, 6-9 July 2004.
- [28] Villapol M.E. and Billington J. *Modelling and Initial Analysis of the Resource Reservation Protocol using Coloured Petri Nets*, Proceedings of the Workshop on Practical Use of High-Level Petri Nets, Aarhus, Denmark, June 27, 2000, pp 91-110.
- [29] Villapol M.E. and Billington J. *Generation of a Service Language for the Resource Reservation Protocol Using Formal Methods*, Proceedings of Eleventh Annual International Symposium of the International Council On Systems Engineering (INCOSE), 1-5 July 2001, on CD-ROM.
- [30] Villapol M.E. *Modelling and Analysis of the Resource Reservation Protocol Using Coloured Petri Nets*. PhD Thesis, University of South Australia, November 2003.
- [31] Villapol M.E. and Billington. *Analysing Properties of the Resource Reservation Protocol*. Proceedings of 24th International Conference on Application and Theory of Petri Nets, June 23-27, 2003, Eindhoven, The Netherlands, LNCS 2679, pp 377-396, Springer-Verlag, 2003.