Generation of a Service Language for the Resource Reservation Protocol Using Formal Methods

María E. Villapol and Jonathan Billington
Cooperative Research Centre for Satellite Systems
University of South Australia
SPRI Building, Mawson Lakes, Adelaide SA 5095
Tel: 08 8302 3371 Fax: 08 8302 3873
maria@spri.levels.unisa.edu.au and jonathan.billington@unisa.edu.au

Abstract. The *Resource Reservation Protocol (RSVP)* is a signalling protocol, which transports and maintains *Quality of Service (QoS)* information along the path of a data flow. It is being modelled and analysed using a verification methodology proposed by (Billington et al 1986). The methodology includes a service and protocol specification. This paper is focused on the definition, modelling, and analysis of the RSVP service specification. A service language including all the possible service primitives sequences was also generated. It is being used as part of the verification process of RSVP. Also, this service specification will allow other resource reservation protocols to be developed that satisfy this service.

INTRODUCTION

Much research and protocol development work has been done recently on the provision of services with *Quality of Service (QoS)* guarantees over the Internet. The *Resource Reservation Protocol (RSVP)* is one of IETF's proposals (Braden et al. 1997) (Durham et al. 1995) for conveying the QoS related information along the communication path of a data flow (where the data flow is a sequence of packets flowing between a source application and one or more destinations).

The Internet Protocols are specified in documents called *Request for Comments (RFCs)*. The standard-related RFCs provide a narrative description of the protocols. In most of these RFCs, there is little, if any, use of formal techniques for specifying communication protocols, such as state tables. Thus, those documents are sometimes ambiguous, difficult to understand, and imprecise. Also, protocol implementations are probably the only mechanism to validate and to "verify" the proposed standards. The cost for fixing errors in the protocol found in the implementation can be high.

In order to verify or design a protocol, there needs to be a statement defining the requirements of the protocol. This is known as a *service specification* (ITU-T 1994). The Internet protocol standards are weak in this respect, as they rarely describe a protocol service.

RSVP is an Internet Standard specified in RFC 2205 (Braden et al. 1997). RFC 2205 has the drawbacks described previously. Those disadvantages together with the fact that RSVP is complex make it a good target for formal specification and verification activities.

RSVP has been modelled and analysed (Villapol et al. 2000) based on a methodology for specifying and validating communication protocols proposed by (Billington et al. 1986). It has already been successfully used in the formal studies of other communication protocols, such as the *OSI Transport protocol Class 0* (Billington et al. 1986), and more recently in the *Wireless Transaction Protocol (WTP)* (Gordon et al. 2000). In the methodology, the specification process is divided into: the service specification and protocol specification. (Villapol et al. 2000) describes the initial RSVP specification, whereas this paper is focused on the service. It provides a definition, modelling, and analysis of the RSVP Service Specification using *Coloured Petri Nets (CPNs)* (Jensen 1997a) and *Finite State Automata (FSA)* (Barret et al. 1979).

The paper is organised as follows. RSVP is described briefly in the next section. Then, the proposed service specification for RSVP is presented. After that, the CPN model of the proposed service specification is outlined briefly. Then the model is analysed. Next, the process for generating the service language generated from the model is described. Finally, the conclusions of the paper are presented. The reader is assumed to be familiar with Coloured Petri Nets (Jensen vol. 1 1997).

OVERVIEW OF RSVP

RSVP is a signalling protocol developed to create and to maintain QoS information in the form of resource reservations on each node along the path of a data flow. RSVP may run on each end host running a QoS application and router (ie RSVP-aware router). It reserves resources for a data flow from the sender to one or more destinations (ie multicast destination). Unlike other signalling protocols, RSVP destinations (or receivers) initiate resource reservation establishment by sending reservation requests. Those requests travel on the reverse path of the data flow by following the pre-established route setup by RSVP (Braden et al. 1997). RSVP is also responsible for maintaining those reservation states on each node along the way of the data flow. It follows a soft-state approach where the reservation states established at each node must be refreshed periodically; otherwise they will be automatically removed. The approach is intended to deal with IP dynamic route changes, dynamic multicast group membership and dynamic QoS changes (Braden et al. 1997).

RSVP runs on the top of IP[1] at the level of a transport protocol. It only carries control information. Figure 1 illustrates the Internet protocol stack showing the location of RSVP. A description of these protocols, apart from RSVP, is beyond the scope of this paper; however there is a wide range of literature related to the Internet protocols (eg (Comer 2000)).

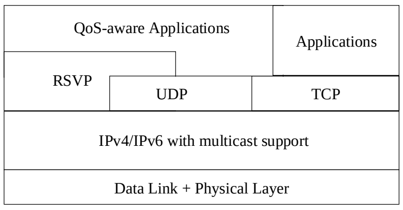

Figure 1: RSVP in the Internet architecture

RSVP also interacts with other components of the QoS Architecture as explained in (Braden et al. 1997) (Durham et al. 1995). A description of the architecture and the relationship with RSVP is beyond the scope of the paper.

RSVP SERVICE PRIMITIVES

Service primitives provide an abstract way to describe the interaction between the RSVP service user (ie QoS-aware application) and RSVP service provider (see figure 2) (ITU-T 1994). 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

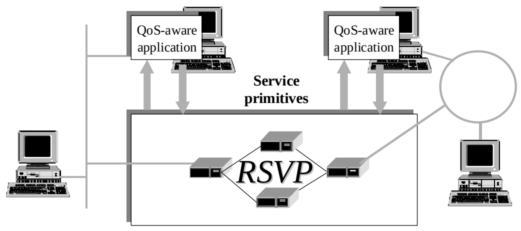

Figure 2: RSVP as a service provider

[1] RSVP may also run on top of UDP as explained in (Braden et al. 1997).

each RSVP-aware router along the path of the data flow interact in order to provide the requested service.

RSVP specification provides several generic interfaces between RSVP and other protocols and mechanisms (Braden et al. 1997). 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 (Braden et al. 1997) and the protocol specification (ie by using *service abstraction*) (Braden et al. 1997) (Villapol et al. 2000).

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:

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 No Defined
RSVP-RelReceiver	Request Indication	RelReserve No 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

1. **RSVP-Sender (Req/Ind):** a sender application uses this primitive to establish the characteristics of a data flow for a RSVP session. A *session* is a data flow with a particular destination and transport-layer protocol and is identified by an IP destination address (unicast or multicast) of the data flow, IP protocol ID, and destination port (optional) (eg UDP/TCP destination port field) (Braden et al. 1997).

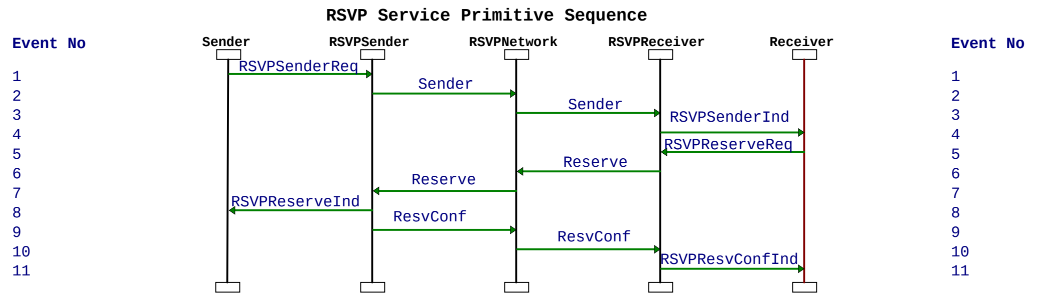

Figure 3: A typical reservation setup sequence

2. RSVP-Reserve (Req/Ind): a receiver application uses this primitive to establish or to modify a reservation for a session.

3. RSVP-RelSender (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.

4. RSVP-RelReceiver (Req/Ind): a receiver application uses this primitive to close its session with the sender. That means that the sender flow travelling to the receiver will not eventually have any QoS reserved.

5. RSVP-ChangeSender (Req/Ind): a sender application uses this primitive to modify the traffic characteristics of the data flow for a session.

6. RSVP-ChangeResv (Req/Ind): a receiver application uses this primitive to modify a reservation for the session. The reservation information can be used by the other end or for the network.

7. RSVP-ResvConf (Ind): is used by the service provider to confirm a reservation.

8. RSVP-SenderError (Ind): is used by the service provider to report an error in propagation or installation of the Sender's data flow characteristics.

9. RSVP-ResvError (Ind): is used by the service provider to report a reservation failure inside the network.

In order to illustrate some of the above definitions, figure 3 shows a typical RSVP service primitive sequence. It shows the "control service units" interchange between the sender, the network and the receiver intended to establish a resource reservation. In there, the Sender and Receiver represent the service users (ie QoS-aware applications), and RSVPSender, RSVPNetwork, and RSVPReceiver represent the service provider. Vertical lines on the picture represent communicating entities. A horizontal line is used to represent the flow of a control service unit generated as result of invocation of a service primitive. Time increases from the top of the chart, to the bottom of the chart.

MODEL OF RSVP SERVICE SPECIFICATION

A model of the RSVP service specification is created with *Coloured Petri Nets (CPNs)* (Jensen vol. 1 1997) using the Design/CPN software tool (Kristensen 1998) (Meta 1993). Coloured Petri Nets are a formal method, which can be used for modelling and analysis of distributed and concurrent systems (Jensen vol. 3 1997).

Assumptions and Requirements. RSVP specification does not provide an explicit definition of the service primitive sequences at each application/RSVP interface. It is required for developing the model. Thus, those sequences were defined based on the description of RSVP (Braden et al. 1997) and represented by using a service primitive sequence table. The table is too big (a 15x15 matrix) to be shown in this paper.

The model of the RSVP Service Specification includes all the service primitives. The model is created based on the information in the service primitive sequence table and the following assumptions, which are intended to simplify the modelling and analysis tasks:

1. Topology: the model includes one sender and one receiver connected by a unicast network which may have multiple routers connected in any topology.

2. Flow/Sessions: a RSVP session is independent of any other. Different sessions may generate the same service primitive sequences, so only one session is considered. For simplicity, only one flow is considered.

3. Errors: only the errors that may occur during the processing of the control service units generated by a RSVPSenderReq, RSVPChangeSenderReq, RSVPReserveReq, and RSVPChangeReserveReq are considered. However, other errors that may be generated by the network as result of RSVP protocol procedures are not.

4. Merging: is a mechanism provided by RSVP to control RSVP message overhead (Braden et al. 1997).

It may affect the sequences of service primitives generated on the sender side. However, merging is not possible given that the network is unicast and only one flow is considered.

5. Reservation Confirms: only an end-to-end confirmation can be generated given that the network is unicast and only one flow is considered.

6. Packet losses: packet loss recovery is a concern of the protocol (ie RSVP). RSVP provides a mechanism to deal with that by using periodic refreshes and cleanups (Braden et al. 1997). Packet loss shouldn't affect the sequences of service primitives.

7. Changes: both the sender and the receiver hosts may generate multiple change requests. The number of change requests, which can be *generated* by an application, is limited in the model. It simplifies the analysis of the model, such as language generation and analysis.

8. Overtaking: the control service units sent from the sender to the receiver or vice versa may not arrive at the destination in the same order they were delivered. The reasons for overtaking are the existence of multiple routes between sender and receiver and IP forwarding approach (Comer 2000).

General structure. In Design/CPN, the model is arranged in pages. A CPN model includes the *states*, which a system may be and the *transitions* between them. Local states are represented by *places* which have a *type (colour set)* associated with. Transitions represent actions or service events. A *marking* shows the state of a CPN and consists of a set of *tokens* distributed on each place. A token is a value which belongs to the type of the place (colour set). Transitions are drawn as rectangles and places as circles or ellipses.

The detailed model of the RSVP Service Specification consists of ten pages (figure 4). The hierarchical view has been designed based on the different request-indication service primitive sequences. The proposed service primitive sequences are: data flow information setup (RSVPSender), reservation setup (RSVPReserve), sender release (RSVPRelSender), receiver release (RSVPRelReceiver), change of data flow information (RSVPChangeSender), change of reservation (RSVPChangeResv), error during data flow information installation (RSVPSenderError), error during reservation installation (RSVPResvError), and confirmation of a reservation request (RSVPResvConf).

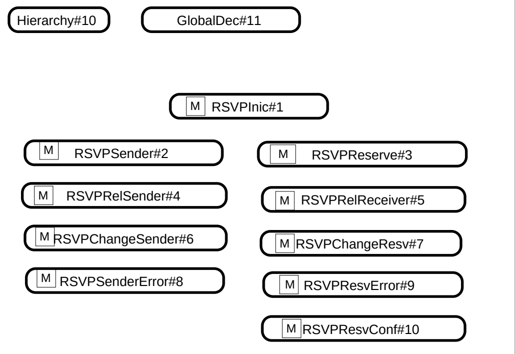

Figure 4: CPN Hierarchy page

```
(* States of RSVP entities *)
color State = with
SESSION|WAITINGRESV|RESVREADY|CLOSED|
NOSENDER|NORECEIVER;
color SenderState = subset State with
[SESSION,WAITINGRESV,RESVREADY,CLOSED,
NORECEIVER];
color ReceiverState = subset State with
[SESSION,WAITINGRESV,RESVREADY,CLOSED,
NOSENDER];
(* Service units *)
color UpStreamMessages = with
Reserve|ChangeResv|SenderError|
RelReceiver;
color DownStreamMessages = with
Sender|ChangeSender|ResvError|RelSender|
ResvConf;
(* User Request *)
color AppReq = with
ChangeSnd|ChangeReserve;
(* Inic *)
color Inic = with START;
var sta: State;
(* Functions *)
fun pathexists(sta:State)= (sta =
WAITINGRESV orelse sta = RESVREADY orelse
sta = NORECEIVER);
fun resvexists (sta:State) = sta =
RESVREADY;
```

Figure 5: Global declaration

Global Declaration. Figure 5 shows the colour sets, variables, and functions from the global declaration node. The colour set *State* indicates the possible states of application/RSVP interface at each end. Two subsets of the colour set State have been defined. The *SenderState* subset indicates the possible states of the Sender/RSVP interface, while the *ReceiverState* indicates the possible states of the Receiver/RSVP interface. The states are defined as follows:

- **CLOSED:** when the sender is in this state, the session is closed. If the receiver is in this state, the receiver application has torn down the existing reservation for the session.
- **SESSION:** is the initial state for both the sender and receiver.
- **WAITINGRESV:** means that the Sender's data flow information has been established but as yet no reservation request has been received.
- **RESVREADY:** means that a reservation has been established over the Sender's data flow information and along the path of the data flow.

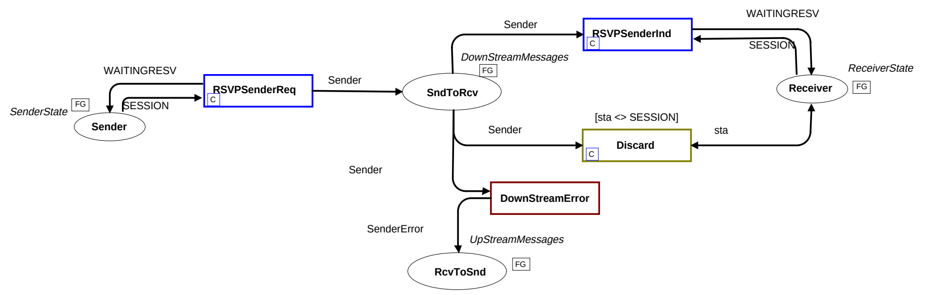

Figure 6: CPN model of RSVP Sender primitive sequence

The following states have been included to control the service primitives sequences:

- **NOSENDER:** the receiver has received an indication that the sender user has left the session.
- **NORECIEVER:** the sender has received an indication that the receiver user has torn down the existing reservation for the data flow and the session.

The colour set *AppReq* represents the number of either sender or reservation changes that can be generated by the sender and receiver, respectively.

There are nine basic control service units represented by the colour sets *UpstreamMessages* and *DowstreamMessages*. The former represents the control service units that travel from the receiver to the sender, while the second represents the ones travelling from sender to receiver. The colour set UpstreamMessages contains the following set of enumerated values:

- **Reserve:** is generated by the receiver protocol entity after it receives a reservation request from the receiver user.
- **ChangeResv:** is generated by the receiver protocol entity after it receives a change reservation request from the receiver user.
- **SenderError:** is generated by the network and indicates that the sender's data flow couldn't be installed for a particular reason.
- **RelReciever:** is generated by the receiver protocol after receiving a release request from the receiver user.

The colour set DowstreamMessages represents the following set of enumerated values:

- **Sender:** is generated by the sender protocol entity to communicate the sender's data flow characteristics along the path of the data flow.
- **ChangeSender:** is generated by the sender protocol entity to change the characteristics of the sender's data flow.
- **ResvError:** is generated by the network as an indication that the reservation couldn't been installed for a particular reason.
- **RelSender:** is generated by the sender protocol after receiving a release request from the sender user.
- **ResvConf:** is generated by the sender as a confirmation of a reservation request.

The variable *sta* is typed by the colour set State. The functions are used to simplify guard inscriptions. A *pathexists* function means the traffic characteristics of the data flow has been established in the correspondent node (ie the RSVP entity – sender or receiver places - is in WAITINGRESV or RESVREADY state). A *resvexists* function means that a reservation has been established (ie the correspondent RSVP entity is in RESVREADY state).

Structure of CPN models. Three types of transition are used to represent the service specification: *service primitive transition, error transition, and discard transition*:

- **Service primitive transition:** represents one of the service primitives in table 1. The name of each transition is the corresponding name in the table.
- **Error transition:** represents an error, which may have happened in the network or in the end host.
- **Discard transition:** cleans the queues after a session has been closed. It also handled situations, which could lead to improper primitive sequences. In addition, it is intended to simplify the analysis of the model against incorrect termination. It represents actions that would be applied for the protocol (ie RSVP) or not.

A detailed description of all the pages of the model is not possible due to space limitation. In order to illustrate the structure and functionality of the model two of the main component pages of the model are described.

Each page of the model may have four places. Two of them represent the state of the Sender application/RSVP interface (ie Sender place) and the Receiver application/RSVP interface (ie Receiver place). The others (ie SndToRcv and RcvToSnd places) represent the communication between the sender and receiver applications. *Fusion places (FG)* are used to group identical places in different pages.

Sender primitive sequence. The CPN model of the sender primitive sequence is shown in figure 6. The transition *RSVPSenderReq* models the action taken by the Sender application to establish the initial traffic

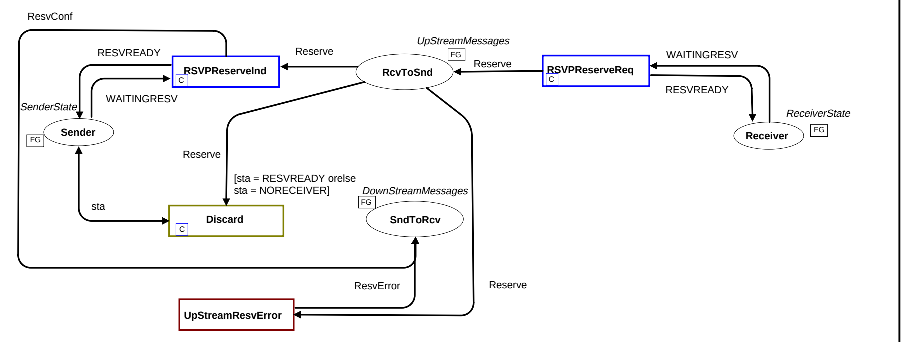

Figure 7: CPN model of RSVP Reserve primitive sequence

characteristics of the sender's data flow. The result of the action is to send a control service unit (ie Sender) to the Receiver application. If there is not any error during the installation of data flow's characteristics, the receiver application will receive a *RSVPSenderInd* as an indication of the existence of a data flow with some particular characteristics from the sender application. Otherwise the network or the receiver may have found problems to establish the characteristics of the data flow, so a *DownStreamError* will occur and a *SenderError* will be generated and transmitted back to the sender. The *Discard* transition models any action that should be taken by the protocol to meet the service specification. For example, the receiver application can not receive an indication if it is in any state different from the initial state (ie SESSION state).

Reserve primitive sequence. Figure 7 shows the CPN model of the reserve primitive sequence. When a receiver application wishes to make a reservation for the flow whose information has already been installed (ie the receiver is in WAITINGRESV state), it generates a *RSVPReserveReq*. As result of this action a Reserve service unit will be sent across the network to the Sender. If there is not any error during the establishment of the reservation, the sender application will receive a *RSVPReserveInd* indicating the characteristics of the reservation, which has been established along the path of the data flow and send a reservation confirmation (*ResvConf*). Otherwise, the network or the sender may have found problems installing the reservation, so a *UpStreamError* will occur and a *ResvError* will be generated and sent back to the receiver. If a reservation has already been established (the sender is in either RESVREADY or NORECEIVER state), the Reserve is discarded by the *Discard* transition. It also may be noted that if the sender is in a SESSION or CLOSED state, it is because there is no current information about the characteristics of the data flow, so a *Reserve* service unit is discarded and an error message is sent back to the sender.

Initial state. The Sender and receiver application begin in the SESSION state – the Sender and Receiver places have an initial marking of 1`SESSION. The communication places (ie SndToRcv and RcvToSnd) are empty. The application places, SenderApp and ReceiverApp, have only one token each indicating that a Sender Change and Reservation Change may occur. The *RSVPInic* page initialises the model according to this initial marking.

ANALYSIS OF THE CPN MODEL

The model was verified against the basic behavioural properties by using a well-known technique called *state space analysis* (Jensen vol. 2 1997). The state space or *OCC graph* is a directed graph which contains all possible occurrence sequences (represented by nodes) and all reachable markings (represented by arcs). A state space generator is provided in Design/CPN. Thus, the full state report for the entire state space was generated. It has *880* nodes and *3391* arcs.

The state report also shows the home and liveness properties (Jensen vol. 1 1997). There are four dead markings (or terminal states). They correspond to the states where the sender has finished a session, the receiver has torn down any existing reservation and all service units have been removed from the communication places. Those states are the same except for the value of the SenderApp and ReceiverApp places (ie application places) which may or may not have a token. The marking value of those places depends on when a sender or receiver release occurs (before or after a change sender or change reservation). Thus, for the initial marking, the service behaves as expected (ie no deadlocks). Since those markings form a home space the service always terminates as expected. A *home space* is a set of markings that can always be reached from all reachable markings (Jensen vol. 1 1997).

The state report also provides information about boundednees of places. Thus, it was checked that the communication places are bounded, so a limited numbers a control service units are generated. The number of tokens located in the state places (Sender and Receiver) is always one. Both can be in all the expected states (eg WAITINGRESV state). It means, for example, that an end-to-end sender state or

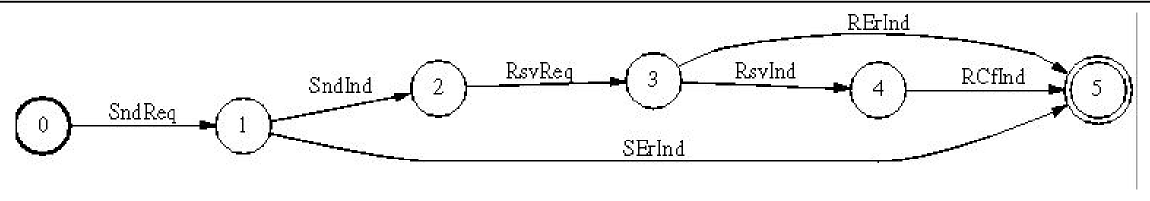

Figure 8: Minimal FSA for reservation setup service primitive sequences

reservation can be established. Those results are expected.

SERVICE LANGUAGE

The service language defines all the possible service primitive sequences at both service access points. The OCC graph includes not only the transitions representing RSVP service primitives but other transitions, such as error transitions. Design/CPN does not provide explicit support for language generation, which includes only the service primitive sequences. Since the OCC graph can be seen as a *Finite State Automaton (FSA)*, the service language was generated by using a well-known FSA reduction technique (Barret et al. 1979) with the aid of the FSM tool[2].

The following steps were followed for the service language generation. The service primitive transitions in the CPN model were assigned different numbers (no zero). The others were marked as zero epsilon (or empty) transitions. A program wrote the OCCgraph into a file format accepted by FSM[3].

Then, the algorithm described in (Barret et al. 1979) was used to generate the minimal FSA. The algorithm is based on the following steps: removal of empty move cycles (remove empties), removal of empty moves (remove empties), removal of non determinism (determinisation), removal of inaccessible states (minimisation), and reduction by identifying and merging equivalent states (minimisation). The FSM tool provides all the programs for supporting those steps.

Table 2 shows the size of the reduced FSA (minimal FSA) versus the size of the OCC graph in terms of the number of nodes and arcs and the number of terminal states. It also shows the number of service primitive sequences (ie service language), which can be accepted by the FSA.

	OCC graph	Minimal FSA
# Nodes	880	106
# Arcs	3391	434
# Final States	4	1
# Service Primitive Sequences	-	20406

Table 2: OCC graph vs minimal FSA

[2] see *http://www.research.att.com/sw/tools/fsm.*
[3] see *http://www.research.att.com/sw/tools/fsm/doc.*

Visual inspection of the minimal FSA is difficult given the size of the automata and the number of sequences accepted. In order to increase confidence that the service specification is behaving as expected, partial analysis was carried out by generating the FSA for only some sequences. For example, figure 8 shows the FSA for the reservation setup sequences where change and release service primitives have not been taken into account. The only terminal state is represented by a double circle in the FSA. A service primitive sequence starts in node 0 and finishes in node 5 (ie terminal node). The following sequences can be found in the FSA:

1. RSVPSenderReq(SndReq), RSVPSenderInd(SndInd), RSVPReserveReq(RsvReq), RSVPResvErrorInd (RerInd);
2. RSVPSenderReq(SndReq), RSVPSenderInd(SndInd), RSVPReserveReq(RsvReq), RSVPReserveInd(RsvInd), RSVPResvConf(RCfInd);
3. RSVPSenderReq(SndReq), RSVPSenderErrorInd(SErInd).

They correspond to the intuition what should happen.

CONCLUSIONS

In this paper, a service specification for RSVP has been presented. The proposed service specification was also modelled using Coloured Petri Nets. Initial analysis of the model showed that it worked as expected. This analysis was based on the study of behavioural properties and the use of a well-known analysis technique called state space exploration. The state space or OCC graph was employed to generate the service language. Since Design/CPN does not support explicit generation of the language, the FSM tool was used instead. It required a simple conversion from the state space to the file format accepted by FSM. Then, the FSA was reduced using a well-known reduction technique. Since, inspection of all the sequences accepted by the minimal FSA is difficult given the size of the service language, partial analyses were carried out.

Future work includes comparing the service specification presented in this paper with the protocol specification in (Villapol et al. 2000).

Finally, the proposed service specification will allow other resource reservation protocols to be developed that satisfy this service.

ACKNOWLEDGMENT

This work was carried out with financial support from the Commonwealth of Australia through the Cooperative Research Centres Program.

REFERENCES

Barret W. and Couch J. *Compiler construction: theory and practice*. Science Research Associates, Chicago, 1979.

Billington J. and M.C. Wilbur-Ham. "Automated Protocol Verification". *Protocol Specification, Testing, and Verification*. M. Diaz (editor). Elsevier Science Publisher, 1986 pp 59-70.

Braden R., et al. "Resource Reservation Protocol (RSVP) -- Version 1: Functional Specification". RFC 2205, IETF, September, 1997.

Comer D. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. Vol. 1, Prentice Hall, 4th Edition, 2000.

Durham D. and Yavatkar R. *Inside the Internet's Resource Reservation Protocol.* Wiley, USA, 1999.

Gordon S. and Billington J. "Analysing the WAP Class 2 Wireless Transaction Protocol Using Coloured Petri Nets". *Proceedings of 21st International Conference, ICATPN 2000*, Aarhus, Denmark, June 2000, pp 207-226.

ITU-T "Convention for the Definition of OSI Services". *Recommendation X.210*. 1994.

Jensen K. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Vol. 1,2 , and 3, Springer-Verlag, April, 1997.

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.

Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*, Version 2, Meta Software Corporation, Cambridge, 1993.

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.

BIOGRAPHIES

Maria Elena Villapol completed a Graduate study in Computer Science (with Magna Cum Laude Mention) from *Universidad Central de Venezuela* (Venezuela) in 1991. From 1991 to 1992, she worked in one of the four Venezuelan Petroleum Companies (LAGOVEN) as a System Analyst in the Department of Information Technology. In 1992, she joined to *Universidad Central de Venezuela* as a full-time lecturer and researcher. In 1996, she completed a Master of Computer Science from this University. In 1996, she obtained a scholarship from the same University to study in Australia. In 1998, she finished a Master of Digital Communication from Monash University (Australia). Currently, she is a PhD candidate at the University of South Australia, within the Cooperative Research Centre for Satellite Systems. Her research is based on the Modelling and Analysis of the Resource Reservation Protocol (RSVP) using formal methods.

Jonathan Billington has B.E. and MEngSc degrees from Monash University, Australia and a PhD from the University of Cambridge,UK. After working for Adelaide University, he spent 15 years with Telecom Australia Research Laboratories, where he led a team developing protocol engineering tools and techniques. Jonathan is Professor of Computer Systems Engineering at the University of South Australia and the Director of the Computer Systems Engineering Centre, where he leads a group researching distributed and concurrent systems. He has consulted to various companies and government agencies and is currently editor of ISO/IEC 15909 on High-level Petri nets.