

Chapter 2

Non-hierarchical Coloured Petri Nets

This chapter introduces the concepts of non-hierarchical Coloured Petri Nets. This is done by means of a running example consisting of a set of simple communication protocols. Protocols are used because they are easy to explain and understand, and because they involve concurrency, non-determinism, communication, and synchronisation which are key characteristics of concurrent systems. No preliminary knowledge of protocols is assumed.

Section 2.1 introduces the protocol used as a running example. Sections 2.2 and 2.3 introduce the net structure, inscriptions, and enabling and occurrence of transitions using a first model of the protocol. Sections 2.4–2.6 introduce concurrency, conflicts, and guards using a more elaborate model of the protocol. Section 2.7 discusses interactive and automatic simulation of CPN models.

2.1 A Simple Example Protocol

We consider a simple protocol from the transport layer of the Open Systems Interconnection (OSI) reference model [100]. The transport layer is concerned with protocols ensuring reliable transmission between hosts. The protocol is simple and unsophisticated, yet complex enough to illustrate the basic CPN constructs.

The simple protocol consists of a *sender* transferring a number of *data packets* to a *receiver*. Communication takes place over an unreliable network, i.e., packets may be lost and overtaking is possible. The protocol uses sequence numbers, acknowledgements, and retransmissions to ensure that the data packets are delivered once and only once and in the correct order at the receiving end. The protocol deploys a stop-and-wait strategy, i.e., the same data packet is repeatedly retransmitted until a corresponding acknowledgement is received. A data packet consists of a sequence number and the data payload. An acknowledgement consists of a sequence number specifying the number of the next data packet expected by the receiver.

We start with a first, very simple model of the protocol where retransmissions and the unreliability of the network are ignored. The model is then gradually refined

to introduce more and more aspects, including loss of packets on the network. The gradual refinement of the model is used to illustrate the various facilities in the CPN modelling language. When constructing CPN models or formal specifications in general, it is good practice to start by making an initial simple model, omitting certain parts of the system or making simplifying assumptions. The CPN model is then gradually refined and extended to lift the assumptions and add the omitted parts of the system.

2.2 Net Structure and Inscriptions

A CPN model is always created as a graphical drawing and Fig. 2.1 contains a first model of the simple protocol. The left part models the sender, the middle part models the network, and the right part models the receiver. The CPN model contains seven *places*, drawn as ellipses or circles, five *transitions* drawn as rectangular boxes, a number of directed *arcs* connecting places and transitions, and finally some textual *inscriptions* next to the places, transitions, and arcs. The inscriptions are written in the CPN ML programming language. Places and transitions are called *nodes*. Together with the directed arcs they constitute the *net structure*. An arc always connects a place to a transition or a transition to a place. It is illegal to have an arc between two nodes of the same kind, i.e., between two places or two transitions.

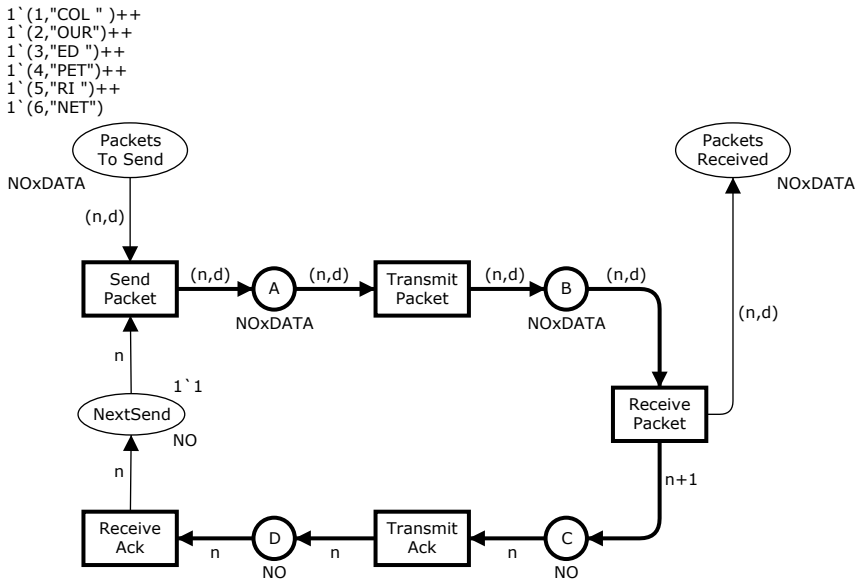


Fig. 2.1 First CPN model of the simple protocol

The places are used to represent the state of the modelled system. Each place can be marked with one or more *tokens*, and each token has a data value attached to it. This data value is called the *token colour*. It is the number of tokens and the token colours on the individual places which together represent the state of the system. This is called a *marking* of the CPN model: the tokens on a specific place constitute the marking of that place. By convention, we write the names of the places inside the ellipses. The names have no formal meaning – but they have huge practical importance for the readability of a CPN model, just like the use of mnemonic names in traditional programming. A similar remark applies to the graphical appearance of the nodes and arcs, i.e., the line thickness, size, colour, and position. The state of the sender is modelled by the two places `PacketsToSend` and `NextSend`. The state of the receiver is modelled by the place `PacketsReceived` and the state of the network is modelled by the places `A`, `B`, `C`, and `D`.

Next to each place is an inscription which determines the set of token colours (data values) that the tokens on that place are allowed to have. The set of possible token colours is specified by means of a type, as known from programming languages, and it is called the *colour set* of the place. By convention, the colour set is written below the place. The places `NextSend`, `C`, and `D` have the colour set `NO`. Colour sets are defined using the CPN ML keyword `colset`, and the colour set `NO` is defined to be equal to the set of all integers `int`:

```
colset NO = int;
```

This means that tokens residing on the three places `NextSend`, `C`, and `D` will have an integer as their token colour. The colour set `NO` is used to model the sequence numbers in the protocol. The remaining four places have the colour set `NOxDATA`, which is defined to be the product of the types `NO` and `DATA`. This type contains all two-tuples (pairs) where the first element is an integer and the second element is a text string. Tuples are written using brackets (and) around a comma-separated list. The colour sets are defined as

```
colset DATA = string;
colset NOxDATA = product NO * DATA;
```

The colour set `DATA` is used to model the payload of data packets and is defined to be the set of all text strings `string`. The colour set `NOxDATA` is used to model the data packets, which contain a sequence number and some data.

The inscription on the upper right side of the place `NextSend` specifies that the *initial marking* of this place consists of one token with the token colour (value) 1. Intuitively, this indicates that data packet number 1 is the first data packet to be sent. The inscription on the upper left side of the place `PacketsToSend`:

```
1 ` (1, "COL") ++
1 ` (2, "OUR") ++
1 ` (3, "ED ") ++
1 ` (4, "PET") ++
1 ` (5, "RI ") ++
1 ` (6, "NET")
```

specifies that the initial marking of this place consists of six tokens with the data values

```
(1, "COL" ),
(2, "OUR" ),
(3, "ED  " ),
(4, "PET" ),
(5, "RI  " ),
(6, "NET" ).
```

The symbols ++ and ` are operators used to construct a *multiset* consisting of these six token colours. A multiset is similar to a set, except that values can appear more than once. The infix operator ` takes a non-negative integer as its left argument, specifying the *number of appearances* of the element provided as the right argument. The operator ++ takes two multisets as arguments and returns their union (the sum). The initial marking of PacketsToSend consists of six tokens representing the data packets which are to be transmitted. The initial marking of a place is, by convention, written above the place. The absence of an inscription specifying the initial marking means that the place initially contains no tokens. This is the case for the places A, B, C, D, and PacketsReceived.

The five transitions drawn as rectangles represent the events that can take place in the system. As with places, the names of the transitions are written inside the rectangles. The transition names have no formal meaning, but they are very important for the readability of the model. When a transition *occurs*, it removes tokens from its *input places* (those places that have an arc leading to the transition) and it adds tokens to its *output places* (those places that have an arc coming from the transition). The colours of the tokens that are removed from input places and added to output places when a transition occurs are determined by means of the *arc expressions*, which are the textual inscriptions positioned next to the individual arcs.

The arc expressions are written in the CPN ML programming language and are built from variables, constants, operators, and functions. When all variables in an expression are bound to values of the correct type, the expression can be *evaluated*. As an example, consider the two arc expressions n and (n, d) on the arcs connected to the transition SendPacket. They contain the variables n and d , declared as

```
var n : NO;
var d : DATA;
```

This means that n must be bound to a value of type NO (i.e., an integer), while d must be bound to a value of type DATA (i.e., a text string). We may, for example, consider the *binding* (variable assignment)

```
⟨n=3, d="CPN"⟩
```

which binds n to 3 and d to "CPN". For this binding the arc expressions evaluate to the following values (token colours), where '→' should be read as 'evaluates to':

$$\begin{aligned}
 n &\rightarrow 3 \\
 (n, d) &\rightarrow (3, \text{"CPN"})
 \end{aligned}$$

All arc expressions in the CPN model of the protocol evaluate to a single token colour (i.e., a multiset containing a single token). This means that the occurrence of a transition removes one token from each input place and adds one token to each output place. However, in general, arc expressions may evaluate to a multiset of token colours, and this means that there may be zero, exactly one token, or more than one token removed from an input place or added to an output place. This will be illustrated later with some further examples.

2.3 Enabling and Occurrence of Transitions

Next, consider Fig. 2.2, which shows the protocol model with its initial marking M_0 . The marking of each place is indicated next to the place. The number of tokens on the place is shown in a small circle, and the detailed token colours are indicated in a box positioned next to the small circle. As explained earlier, the initial marking has six tokens on PacketsToSend and one token on NextSend. All other places are unmarked, i.e., have no tokens.

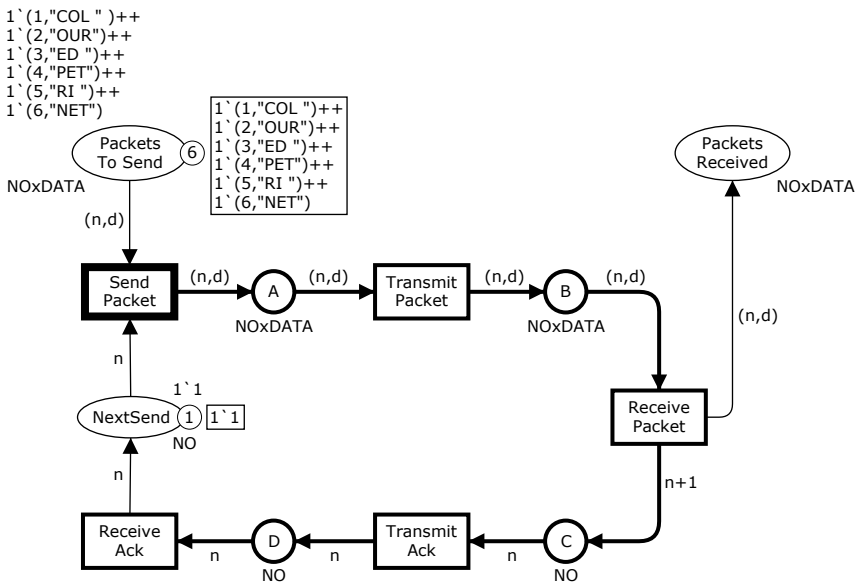


Fig. 2.2 Initial marking M_0

The arc expressions on the input arcs of a transition determine whether the transition is *enabled*, i.e., is able to *occur* in a given marking. For a transition to be enabled, it must be possible to find a binding of the variables that appear in the surrounding arc expressions of the transition such that the arc expression of each input arc evaluates to a multiset of token colours that is present on the corresponding input place. When a transition occurs with a given binding, it removes from each input place the multiset of token colours to which the corresponding input arc expression evaluates. Analogously, it adds to each output place the multiset of token colours to which the corresponding output arc expression evaluates.

Consider now the transition `SendPacket`. In Fig. 2.2, the transition `SendPacket` has a thick border, whereas the other four transitions do not. This indicates that `SendPacket` is the only transition that has an enabled binding in the marking M_0 . The other transitions are disabled because there are no tokens on their input places. When the transition `SendPacket` occurs, it removes a token from each of the input places `NextSend` and `PacketsToSend`. The arc expressions of the two input arcs are n and (n, d) , where n and d (as shown earlier) are declared as

```
var n : NO;
var d : DATA;
```

The initial marking of the place `NextSend` contains a single token with colour 1. This means that the variable n must be bound to 1. Otherwise, the expression on the arc from `NextSend` would evaluate to a token colour which is not present at `NextSend`, implying that the transition is *disabled* for that binding. Consider next the arc expression (n, d) on the input arc from `PacketsToSend`. We have already bound n to 1, and now we are looking for a binding of d such that the arc expression (n, d) will evaluate to one of the six token colours that are present on `PacketsToSend`. Obviously, the only possibility is to bind d to the string "COL". Hence, we conclude that the binding

$$\langle n=1, d="COL" \rangle$$

is the only enabled binding for `SendPacket` in the initial marking. An occurrence of `SendPacket` with this binding removes the token with colour 1 from the input place `NextSend`, removes the token with colour $(1, "COL")$ from the input place `PacketsToSend`, and adds a new token with colour $(1, "COL")$ to the output place A. Intuitively, this represents the sending of the first data packet $(1, "COL")$ to the network. Note that it was the token on `NextSend` that determined the data packet to be sent. The packet $(1, "COL")$ is now at place A, waiting to be transmitted by the network. The new marking M_1 is shown in Fig. 2.3.

In the marking M_1 , `TransmitPacket` is the only enabled transition since the other transitions have no tokens on their input places. Place A has a single token with colour $(1, "COL")$, and hence it is straightforward to conclude that $\langle n=1, d="COL" \rangle$ is the only enabled binding of the transition `TransmitPacket` in M_1 . When the transition occurs in that binding, it removes the token $(1, "COL")$ from A and adds a new token with the same token colour to place B. Intuitively, this corresponds

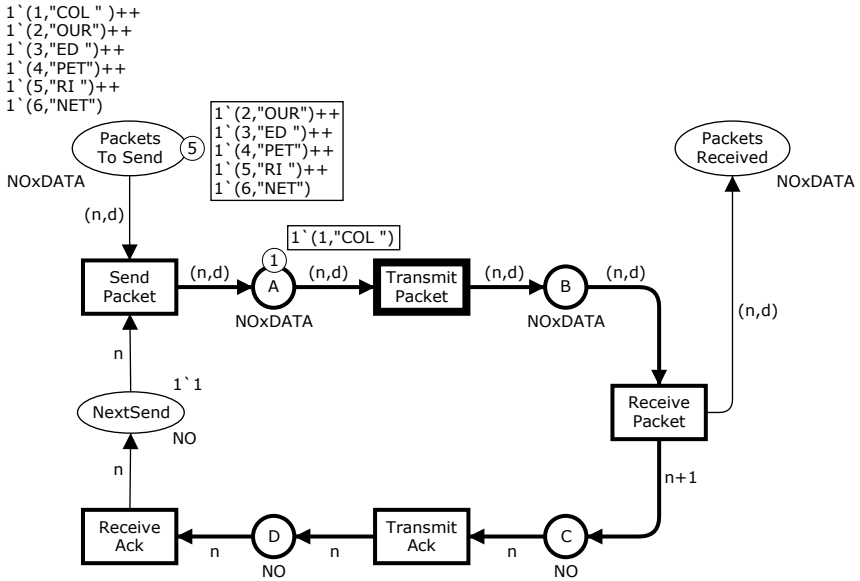


Fig. 2.3 Marking M_1 reached when SendPacket occurs in M_0

to a transmission of data packet number 1 over the network. The data packet is now at place B, waiting to be received. The new marking M_2 is shown in Fig. 2.4.

In the marking M_2 , we have a single enabled transition, ReceivePacket, and once more we use the binding $\langle n=1, d="COL" \rangle$. The occurrence of the transition removes the token with colour $(1, "COL ")$ from place B, adds a token with colour $(1, "COL ")$ to the place PacketsReceived, and adds a token with colour 2 to the place C. The token colour at C becomes 2, since the arc expression $n+1$ on the arc from ReceivePacket to C evaluates to 2 in the above binding. Intuitively, this corresponds to the receipt of data packet number 1 by the receiver. The received data packet is stored in the place PacketsReceived. The token on C represents an acknowledgement sent from the receiver to the sender in order to confirm the receipt of data packet number 1 and to request data packet number 2. The new marking M_3 is shown in Fig. 2.5.

In the marking M_3 there is a single enabled transition TransmitAck. This time we use the binding $\langle n=2 \rangle$. Intuitively, this represents the transmission over the network of the acknowledgement requesting data packet number 2. The new marking M_4 is shown in Fig. 2.6. In the marking M_4 there is a single enabled transition ReceiveAck, and once more we use the binding $\langle n=2 \rangle$. The new marking M_5 is shown in Fig. 2.7. This marking represents a state where the sender is ready to send data packet number 2 (since the first data packet is now known to have been successfully received).

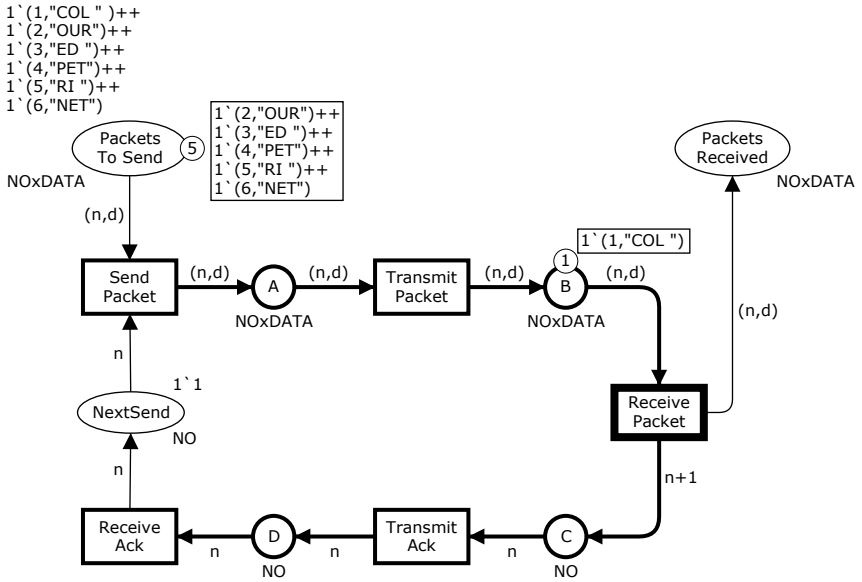


Fig. 2.4 Marking M_2 reached when TransmitPacket occurs in M_1

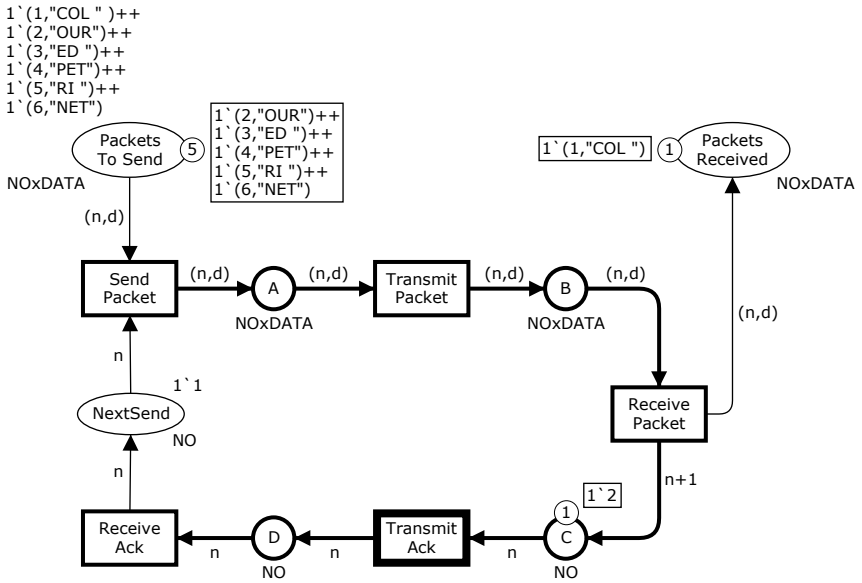


Fig. 2.5 Marking M_3 reached when ReceivePacket occurs in M_2

In the above, we have described the sending, transmission, and reception of data packet number 1 and the corresponding acknowledgement. In the CPN model this

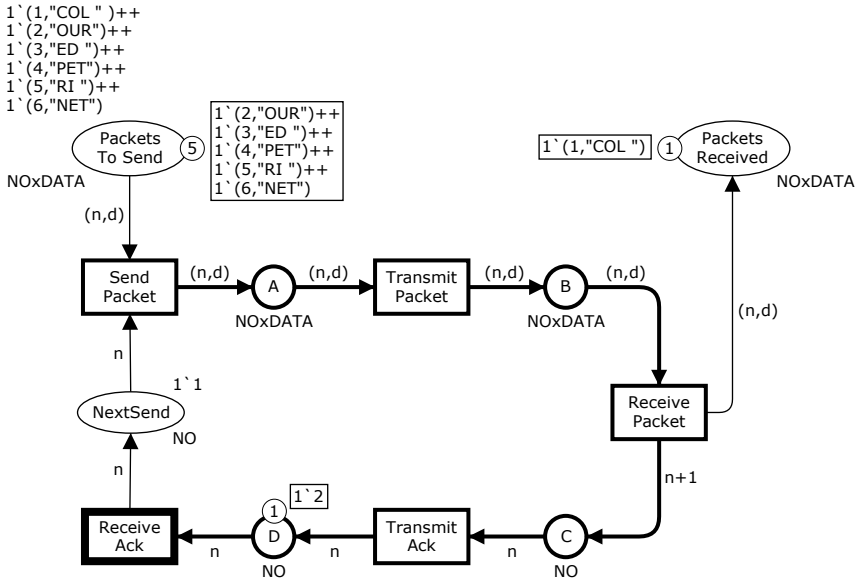


Fig. 2.6 Marking M_4 reached when TransmitAck occurs in M_3

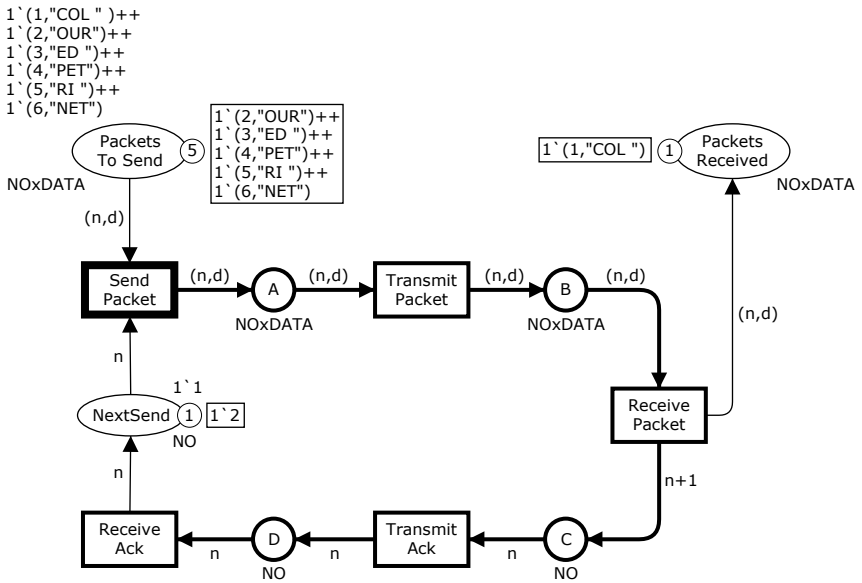


Fig. 2.7 Marking M_5 reached when ReceiveAck occurs in M_4

corresponds to five steps, where each step is the occurrence of a transition in an enabled binding. We have listed these five steps below, where each step is written as a pair consisting of a transition and the occurring binding of the transition. Such a pair is called a *binding element*.

Step	Binding element
1	(SendPacket, $\langle n=1, d="COL" \rangle$)
2	(TransmitPacket, $\langle n=1, d="COL" \rangle$)
3	(ReceivePacket, $\langle n=1, d="COL" \rangle$)
4	(TransmitAck, $\langle n=2 \rangle$)
5	(ReceiveAck, $\langle n=2 \rangle$)

It is easy to see that the next five steps will be similar to the first five steps, except that they describe the sending, transmission, and reception of data packet number 2 and the corresponding acknowledgement:

Step	Binding element
6	(SendPacket, $\langle n=2, d="OUR" \rangle$)
7	(TransmitPacket, $\langle n=2, d="OUR" \rangle$)
8	(ReceivePacket, $\langle n=2, d="OUR" \rangle$)
9	(TransmitAck, $\langle n=3 \rangle$)
10	(ReceiveAck, $\langle n=3 \rangle$)

After these additional five steps, we reach the marking M_{10} shown in Fig. 2.8. Next, we shall have five steps for data packet number 3 and its acknowledgement. Then five steps for data packet 4, five for data packet number 5, and finally five steps for data packet number 6. After these steps the marking M_{30} shown in Fig. 2.9 is reached. This marking corresponds to a state of the protocol where all data packets have been received by the receiver, all acknowledgements have been received by the sender, and no packets are outstanding on the network. This marking has no enabled transitions, and hence it is said to be a *dead marking*.

This completes the survey of the first very simple CPN model of the protocol. This model is *deterministic*, in the sense that each marking reached has exactly one enabled transition with exactly one enabled binding, except for the last marking which is a dead marking. Hence, there is only one possible occurrence sequence, consisting of the markings $M_0, M_1, M_2, \dots, M_{30}$ and the 30 steps described above. It should be noted that this is quite unusual for CPN models, which are usually non-deterministic, i.e., they describe systems where several transitions and bindings are enabled in the same marking.

2.4 Second Model of the Protocol

We now consider a slightly more complex CPN model of the protocol. It is based on the CPN model which was investigated in the previous sections, but now overtaking and the possibility of losing data packets and acknowledgements when they are transmitted over the network are taken into account. Hence, it is necessary to be able to retransmit data packets, and the receiver must check whether it is the expected data packet that arrives. Since acknowledgement may overtake each other, we also have to take into account that the sender may receive acknowledgements out of order. This second model of the protocol is *non-deterministic* and will be used to introduce concurrency and conflict, which are two key concepts for CPN models and other models of concurrency.

Figure 2.10 shows the second CPN model of the protocol in the initial marking M_0 . It has the same five transitions as for the first CPN model of the protocol. We also find six of the places used in the previous model, together with two new places. The place `DataReceived` is used instead of `PacketsReceived`. Now we want to keep only the data from the data packets, not the entire data packets. Hence the colour set of the place `DataReceived` is specified to be `DATA` instead of `NOxDATA`. This place has an initial marking, which consists of one token with colour "" which is the empty text string. The place `NextRec` has the same colour set as the place `NextSend` and it plays a similar role. It contains the number of the data packet that the receiver expects to receive next. This time a small amount of space has been saved in the drawing by specifying the initial marking of the place `PacketsToSend` by means of a symbolic *constant*, defined as

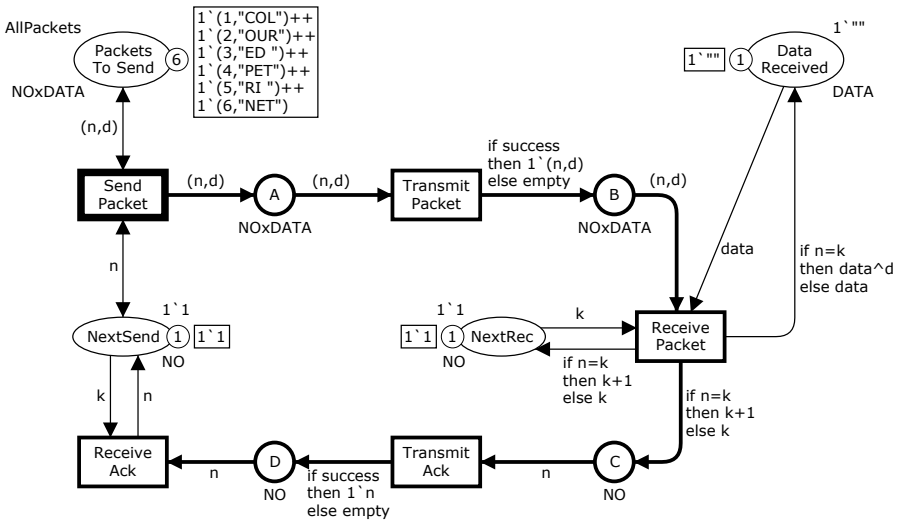


Fig. 2.10 Second CPN model of the protocol in the initial marking M_0

```

val AllPackets = 1 ` (1, "COL") ++ 1 ` (2, "OUR") ++
                 1 ` (3, "ED ") ++ 1 ` (4, "PET") ++
                 1 ` (5, "RI ") ++ 1 ` (6, "NET") ;

```

Consider now the individual transitions. The transition `SendPacket` has the same surrounding arc expressions as before, but now the two input arcs are replaced by *double-headed arcs*. A double-headed arc is a shorthand for the situation where there are two oppositely directed arcs between a place and a transition sharing the same arc expression. This implies that the place is both an input place and an output place for the transition. When the transition occurs with an enabled binding, tokens are removed from the place according to the result of evaluating the arc expression, but they are immediately replaced by new tokens with the same token colours. This means that the marking of the place does not change when the transition occurs, but it does determine the enabling of the transition. In the initial marking, the only enabled transition is `SendPacket` with the binding $\langle n=1, d="COL" \rangle$. As before, an occurrence of `SendPacket` with this binding adds a token to place A representing a data packet to be transmitted over the network. However, now the data packet is not removed from `PacketsToSend` and also the token at `NextSend` is left unchanged. This will allow retransmission of the packet, if this becomes necessary. Figure 2.11 shows the marking M_1 reached when the above binding element occurs in the initial marking.

Consider the marking M_1 and the transition `TransmitPacket`. This transition has the same input arc expression as before, but now there is an additional boolean variable `success`, declared as

```
var success : BOOL;
```

which appears on the output arc. The colour set `BOOL` is defined as

```
colset BOOL = bool;
```

The transition `TransmitPacket` is enabled with two different bindings in M_1 :

```

 $b^+ = \langle n=1, d="COL", success=true \rangle$ 
 $b^- = \langle n=1, d="COL", success=false \rangle$ 

```

The first of these bindings, b^+ , represents a successful transmission over the network. When it occurs, the following happens:

- The data packet $(1, "COL")$ is removed from the input place A.
- A new token representing the same data packet is added to the output place B (in the *if-then-else* expression, the condition `success` evaluates to `true`, while $1 \ (n, d)$ evaluates to $1 \ (1, "COL")$).

Figure 2.12 shows the marking M_2^+ , which is the result of an occurrence of the binding b^+ in M_1 . The second binding, b^- , represents an unsuccessful transmission, i.e., the data packet is lost on the network. When this binding occurs, the following happens:

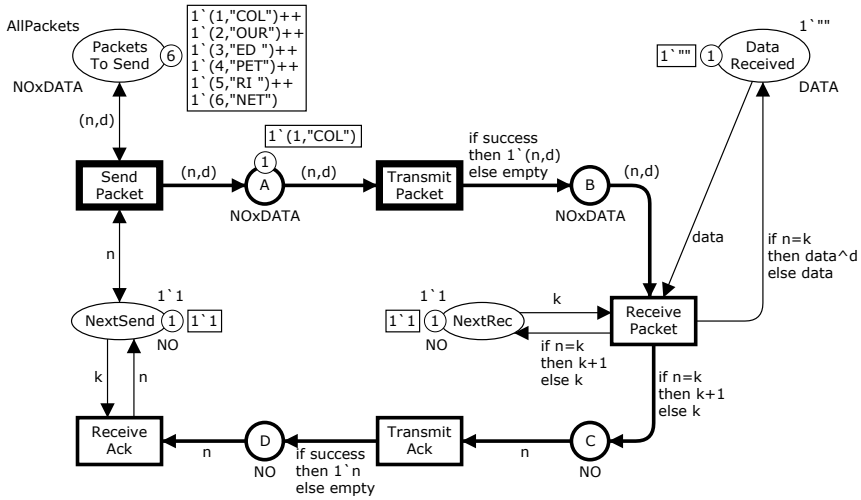


Fig. 2.11 Marking M_1 reached when SendPacket occurs in M_0

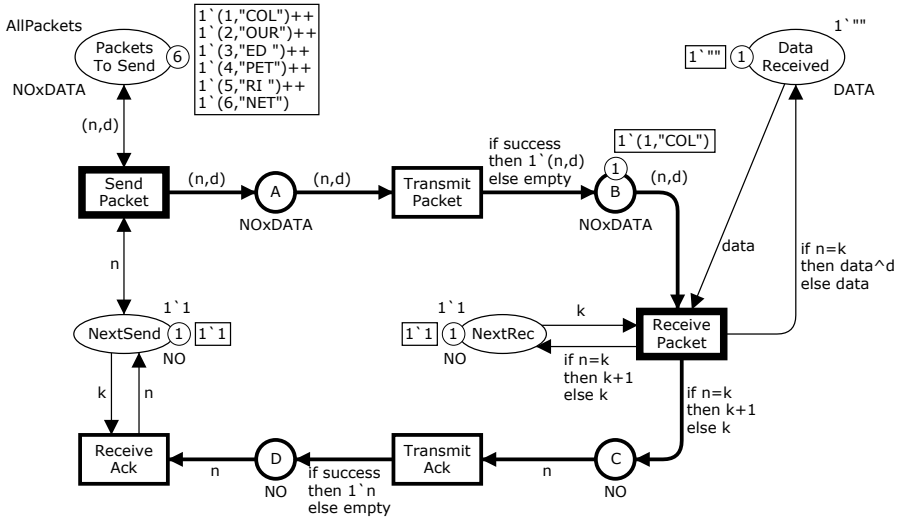


Fig. 2.12 Marking M_2^+ after successful transmission in M_1

- The data packet $(1, "COL")$ is removed from the input place A.
- No token is added to the output place B (in the *if-then-else* expression, the condition *success* evaluates to false, while the constant *empty* evaluates to the empty multiset).

Figure 2.13 shows the marking M_2^- , which is the result of an occurrence of the binding b^- in M_1 . The marking M_2^- is identical to the initial marking M_0 previously shown in Fig. 2.10.

It should be noted that the output arc expression of `TransmitPacket` uses $1 \setminus (n, d)$ and not just (n, d) in the *if-then-else* expression. Using an arc expression such as

```
if success then (n,d) else empty
```

would result in a type mismatch since the then-part and the else-part have different types. The constant `empty` denotes a multiset of tokens, and hence we also need to specify a multiset of tokens in the other branch of the *if-then-else* expression. Types and expressions are discussed further in Chap. 3.

Consider now the reception of data packets in the marking M_2^+ . The transition `ReceivePacket` has four variables on the surrounding arc expressions, with the following purposes:

- n and d denote the sequence number and the data, respectively, of the incoming data packet. The variables n and d will be bound according to the colour of the data packet to be removed from place `B`.
- k (of colour set `NO`) denotes the expected sequence number of the data packet. It will be bound to the colour of the token on the place `NextRec`.
- `data` (of colour set `DATA`) denotes the data that has already been received. It will be bound to the colour of the token on the place `DataReceived`.

When a data packet is present at place `B` there are two different possibilities. Either $n=k$ evaluates to `true`, which means that the data packet being received is the one that the receiver expects, or $n=k$ evaluates to `false` which means that it

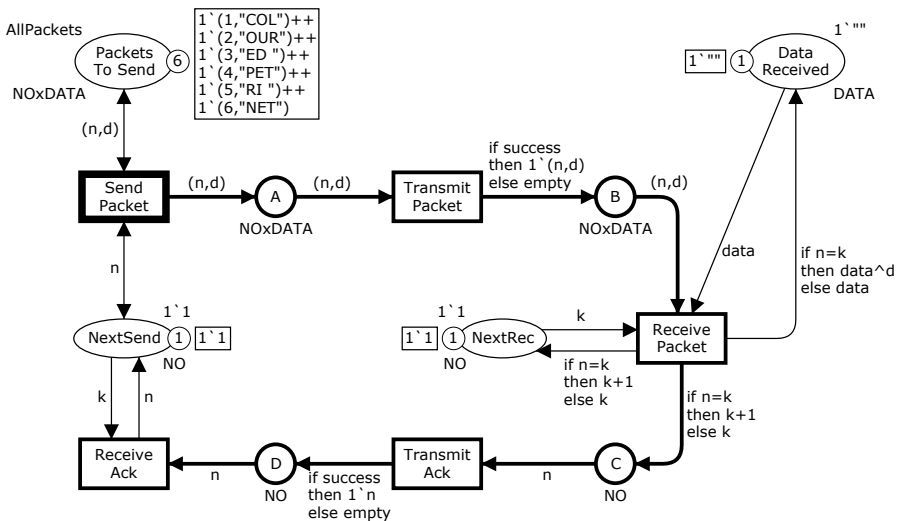


Fig. 2.13 Marking M_2^- after unsuccessful transmission in M_1

is not the data packet expected. If the data packet on place B is the expected data packet (i.e., $n=k$), the following happens:

- The data packet is removed from place B.
- The data in the data packet is concatenated to the end of the data which is already present at the place DataReceived. The operator \wedge is the concatenation operator for text strings.
- The token colour on the place NextRec changes from k to $k+1$, which means that the receiver now waits for the next data packet.
- An acknowledgement is put on place C. The acknowledgement contains the sequence number of the data packet that the receiver is expecting next.

Figure 2.14 shows the result of an occurrence of the transition ReceivePacket in the marking M_2^+ shown in Fig. 2.12. This occurrence of ReceivePacket corresponds to the reception of the expected data packet.

If the data packet on B is not the expected data packet (i.e., $n \neq k$), the following happens:

- The data packet is removed from place B.
- The data in the data packet is ignored (the marking of DataReceived does not change).
- The token colour on the place NextRec does not change, which means that the receiver is waiting for the same data packet as before.
- An acknowledgement is put on place C. The acknowledgement contains the sequence number of the data packet that the receiver is expecting next.

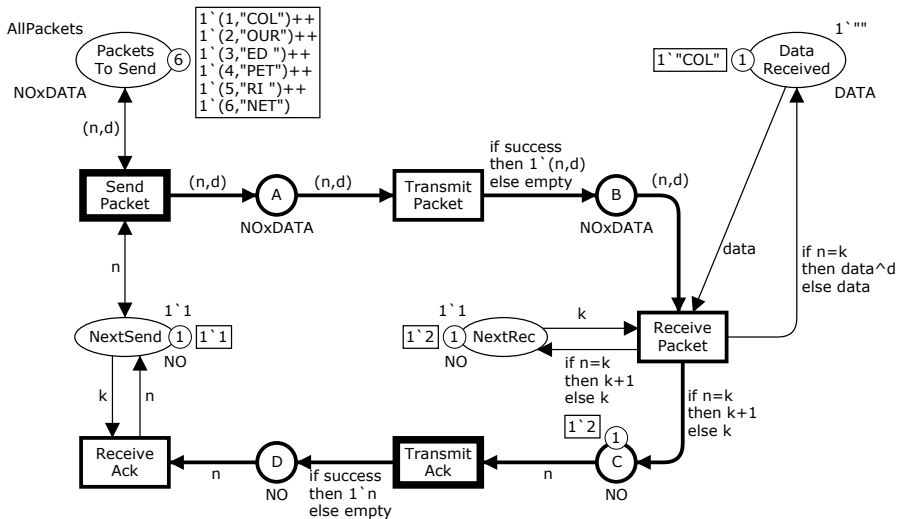


Fig. 2.14 Marking reached when ReceivePacket occurs in M_2^+

The transition `TransmitAck` has a behaviour which is similar to the behaviour of `TransmitPacket`. It removes acknowledgements from place `C` and adds them to the place `D` in case of a successful transmission. The choice is determined by the binding of the variable `success` that appears in the output arc expression.

Consider now the reception of acknowledgements. The transition `ReceiveAck` has two variables:

- `n` (of colour set `NO`) denotes the sequence number in the incoming acknowledgement, and will be bound to the acknowledgement on the place `D`.
- `k` (of colour set `NO`) denotes the sequence number of the data packet which the sender is sending. It will be bound to the colour of the token on the place `NextSend`.

When the transition `ReceiveAck` occurs, it removes an acknowledgement from place `D` and updates the token on `NextSend` to contain the sequence number specified in the acknowledgement. This means that the sender will start sending the data packet that the receiver has requested via the acknowledgement.

2.5 Concurrency and Conflict

We shall now consider the behaviour of the CPN model shown in Fig. 2.10 in further detail. A single binding element is enabled in the initial marking

`(SendPacket, ⟨n=1, d="COL"⟩)`

When it occurs, it leads to the marking M_1 shown in Fig. 2.15 (and Fig 2.11). In the marking M_1 , three different binding elements are enabled:

`SP = (SendPacket, ⟨n=1, d="COL"⟩)`

`TP+ = (TransmitPacket, ⟨n=1, d="COL", success=true⟩)`

`TP- = (TransmitPacket, ⟨n=1, d="COL", success=false⟩)`

The first binding element represents a retransmission of data packet number 1. The second binding element represents a successful transmission of data packet number 1 over the network, and the third binding element represents a transmission where the data packet is lost on the network. The last two binding elements, `TP+` and `TP-`, are in *conflict* with each other. Both of them are enabled, but only one of them can occur since each of them needs a token on place `A`, and there is only one such token in M_1 . However, the binding elements `SP` and `TP+` can occur *concurrently* (i.e., in parallel). To occur, `SP` needs a token on the place `PacketsToSend` and a token on `NextSend`, while `TP+` needs a token on place `A`. This means that the two binding elements use disjoint sets of input tokens, and hence both of them can get the tokens they need without competition or interference with the other binding element. By a similar argument, we can see that `SP` and `TP-` are concurrently enabled. They use disjoint sets of input tokens and hence can occur concurrently.

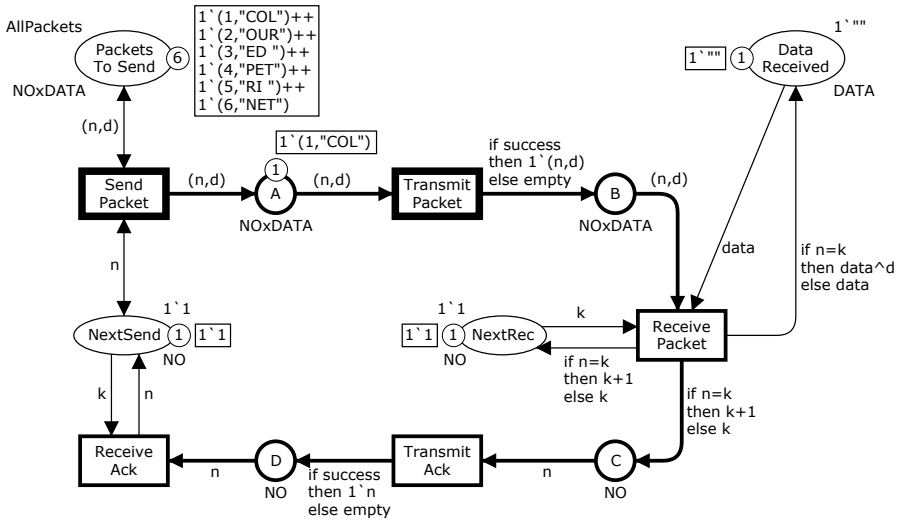


Fig. 2.15 Marking M_1 reached when SendPacket occurs in M_0

Assume that the first and second of the three enabled binding elements in the marking M_1 occur concurrently, i.e., that we have the following *step*, written as a multiset of binding elements:

$$1'(\text{SendPacket}, \langle n=1, d="COL" \rangle) ++ \\ 1'(\text{TransmitPacket}, \langle n=1, d="COL", success=true \rangle)$$

We then reach the marking M_2 shown in Fig. 2.16. In the marking M_2 , we have four enabled binding elements, of which the first three are the same as in the marking M_1 :

$$\begin{aligned} SP &= (\text{SendPacket}, \langle n=1, d="COL" \rangle) \\ TP^+ &= (\text{TransmitPacket}, \langle n=1, d="COL", success=true \rangle) \\ TP^- &= (\text{TransmitPacket}, \langle n=1, d="COL", success=false \rangle) \\ RP &= (\text{ReceivePacket}, \langle n=1, d="COL", k=1, data="" \rangle) \end{aligned}$$

As before, we have a conflict between TP^+ and TP^- , whereas all of the other binding elements are concurrently enabled since they use disjoint multisets of input

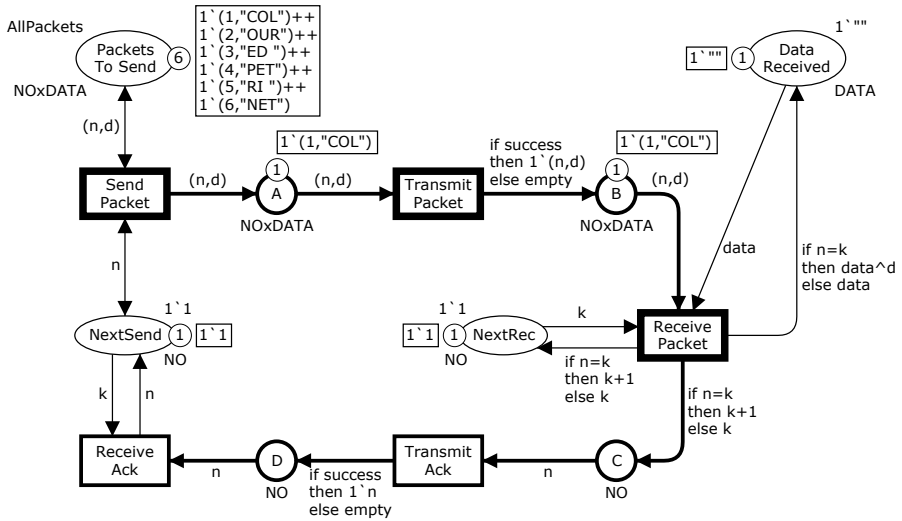


Fig. 2.16 Marking M_2 reached when SendPacket and TransmitPacket occur in M_1

tokens. Let us assume that we have a step where the first and last of the four binding elements occur concurrently, i.e., the following step:

$$1'(\text{SendPacket}, \langle n=1, d="COL" \rangle) ++$$

$$1'(\text{ReceivePacket}, \langle n=1, d="COL", k=1, data="" \rangle)$$

We then reach the marking M_3 shown in Fig. 2.17. In the marking M_3 , we have five enabled binding elements, of which the first three are the same as in the marking M_1 :

$$\text{SP} = (\text{SendPacket}, \langle n=1, "COL" \rangle)$$

$$\text{TP}^+ = (\text{TransmitPacket}, \langle n=1, "COL", \text{success}=\text{true} \rangle)$$

$$\text{TP}^- = (\text{TransmitPacket}, \langle n=1, "COL", \text{success}=\text{false} \rangle)$$

$$\text{TA}^+ = (\text{TransmitAck}, \langle n=2, \text{success}=\text{true} \rangle)$$

$$\text{TA}^- = (\text{TransmitAck}, \langle n=2, \text{success}=\text{false} \rangle)$$

However, this time there are two tokens on place A. This means that TP^+ and TP^- can occur concurrently because there is a token on A for each of the two binding elements. It also means that TP^+ can occur *concurrently with itself*, and the same is true for TP^- . Thus it is possible to transmit multiple packets on the network

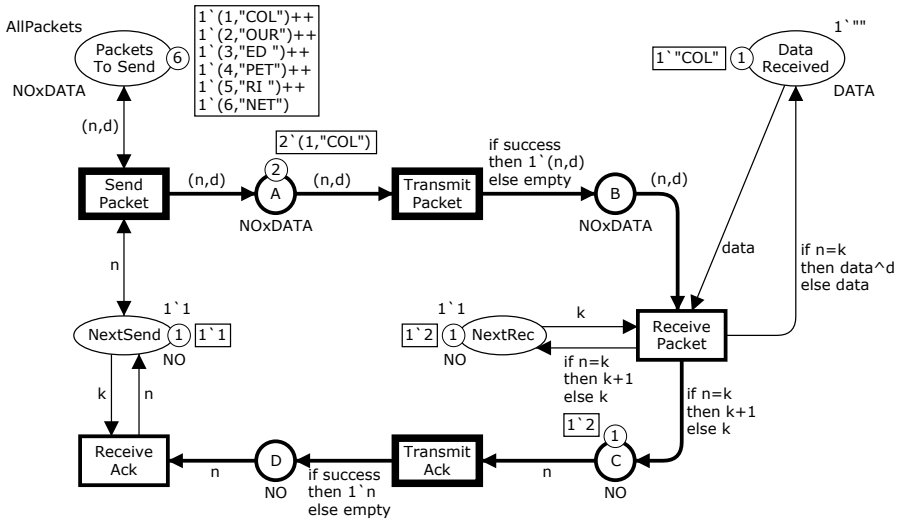


Fig. 2.17 Marking M_3 reached when SendPacket and ReceivePacket occur in M_2

concurrently. Hence, we have the following five enabled steps with bindings for TransmitPacket, where each step is a multiset of binding elements:

- $1' TP^+$,
- $1' TP^-$,
- $1' TP^+ ++ 1' TP^-$,
- $2' TP^+$,
- $2' TP^-$

Moreover, it can be seen that each of the five steps with bindings for TransmitPacket can occur concurrently with the following five steps with bindings for SendPacket and/or TransmitAck:

- $1' SP$,
- $1' TA^+$,
- $1' TA^-$,
- $1' SP ++ 1' TA^+$,
- $1' SP ++ 1' TA^-$.

This means that the marking M_3 has a total of 35 enabled steps (25 for the possible combinations of the individual steps in the two groups above plus 10 because each of the 10 steps constitutes a step on its own).

The above illustrates that it soon becomes complex, time-consuming, and error-prone for human beings to keep track of the enabled binding elements and steps, and the current marking of a CPN model. This is one of the reasons for building and using computer simulators for the execution of CPN models.

A step, in general, consists of a non-empty, finite multiset of concurrently enabled binding elements. A step may consist of a single binding element. An empty multiset

of binding elements is not considered to be a legal step, since it would have no effect and always be enabled. The effect of the occurrence of a set of concurrent binding elements is the sum of the effects caused by the occurrence of the individual binding elements. This means that the marking reached will be the same as that which will be reached if the set of binding elements occur *sequentially*, i.e., one after another in some arbitrary order. As an example, consider the marking M_1 shown in Fig. 2.15 and the enabled step consisting of the following two binding elements:

SP = (SendPacket, $\langle n=1, d="COL" \rangle$)
 TP⁺ = (TransmitPacket, $\langle n=1, d="COL", success=true \rangle$)

The marking M_2 resulting from an occurrence of this step was shown in Fig. 2.16. The marking M_2 is also the marking resulting from an occurrence of SP followed by an occurrence of TP⁺, and it is also the marking resulting from an occurrence of TP⁺ followed by an occurrence of SP. The CPN Tools simulator executes only steps consisting of a single binding element. This is sufficient, since the marking resulting from the occurrence of an enabled step with multiple binding elements is the same as letting the binding elements in the step occur one after another in some arbitrary order. Hence, markings that can be reached via occurrence sequences consisting of steps with multiple binding elements can also be reached via occurrence sequences consisting of steps with a single binding element.

When the first data packet has been sent by an occurrence of SendPacket, we may choose a sequence of binding elements that will successfully transmit the data packet, receive the data packet, successfully transmit the acknowledgement for the data packet, and finally receive the acknowledgement updating the token on NextSend to the value 2:

Step	Binding element
1	(SendPacket, $\langle n=1, d="COL" \rangle$)
2	(TransmitPacket, $\langle n=1, d="COL", success=true \rangle$)
3	(ReceivePacket, $\langle n=1, d="COL", k=1, data="" \rangle$)
4	(TransmitAck, $\langle n=2, success=true \rangle$)
5	(ReceiveAck, $\langle n=2, k=1 \rangle$)

This could be called the successful occurrence sequence for packet number 1. In the successful occurrence sequence, no retransmission of packet number 1 takes place. However, it should be noted that the transition SendPacket is enabled in all of the markings of the successful occurrence sequence. If, in any of these markings, we choose to execute SendPacket, this represents a retransmission of data packet number 1. Intuitively, the retransmission happens because the transitions in the successful occurrence sequence are too slow in occurring and hence are outraced by the second occurrence of SendPacket, i.e., the retransmission of data packet number 1. This means that we have described a time-related behaviour without the explicit use

of time. What is important at the chosen abstraction level is not when a retransmission may occur, but the simple fact that it is possible that such a retransmission can occur. While we are executing the successful occurrence sequence for packet number 1, we may also deviate from it by choosing a binding for `TransmitPacket` or `TransmitAck` which loses the packet/acknowledgement, i.e., a binding in which `success=false`. Then `SendPacket` will be the only enabled transition, and a retransmission will be the only possible way to continue.

The CPN model presented in this section is without any reference to time. It is specified that retransmissions are possible, but we do not specify how long the sender should wait before performing such retransmissions. What matters is the possible sequences in which the various events (binding elements) may occur: at least for the moment, we are uninterested in the durations of and start/end times for the individual events. Timed CP-nets will be introduced in Chap. 10; these make it possible to model the time taken by the various events in the system.

Notice that it is possible to reach markings where place A contains two different tokens, for example, the multiset $1 \setminus (1, \text{"COL"}) ++ 1 \setminus (3, \text{"ED"})$ representing data packets numbers 1 and 3. In this situation the variables `n` and `d` of `TransmitPacket` can be bound such that (n, d) evaluates to $(1, \text{"COL"})$ or $(3, \text{"ED"})$, and hence it is possible for data packet 3 to overtake data packet 1. A similar remark applies to data packets on place B and acknowledgements on places C and D.

2.6 Guards

In the discussion above, we have seen that it is the input arc expressions that determine whether a transition is enabled in a given marking. However, transitions are also allowed to have a *guard*, which is a boolean expression. When a guard is present, it must evaluate to `true` for the binding to be enabled, otherwise the binding is disabled and cannot occur. Hence, a guard puts an extra constraint on the enabling of bindings for a transition. Figure 2.18 shows a variant of the receiver part of the protocol which illustrates the use of guards. In this variant, the reception of data packets, previously modelled by `ReceivePacket`, has been split into two transitions: `DiscardPacket` and `ReceiveNext`. The idea is that `ReceiveNext` models the case where the data packet received is the one expected, whereas `DiscardPacket` models the case where the data packet received is not the one expected. This variant also illustrates a modelling choice concerning the number of transitions in a CPN model.

Each of the two transitions `DiscardPacket` and `ReceiveNext` has a guard, which, by convention, is written in square brackets and positioned next to the transition. The guards of the two transitions compare the sequence number in the incoming data packet on place B with the expected sequence number on the place `NextRec`. The guard of the transition `ReceiveNext` is $[n=k]$ expressing the condition that the sequence number of the incoming data packet bound to `n` must be equal to the expected sequence number bound to `k`. The guard $[n<>k]$ of the transition `DiscardPacket` uses the inequality operator `<>` since this transition is only to be

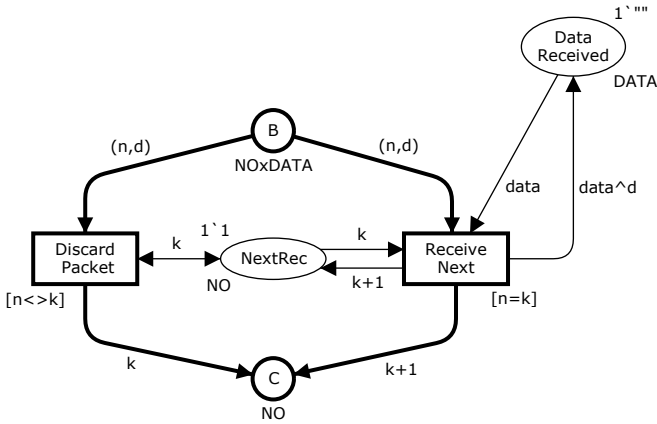


Fig. 2.18 Variant of the receiver part illustrating guards

enabled when the sequence number of the incoming data packet differs from the expected sequence number.

Consider now Fig. 2.19, which depicts a marking where there are two data packets on place B: one corresponding to a data packet that has already been received, and one corresponding to the expected data packet. For this marking, we can consider the following bindings of ReceiveNext:

$$RN_1 = \langle n=1, d="COL", k=2, data="COL" \rangle$$

$$RN_2 = \langle n=2, d="OUR", k=2, data="COL" \rangle$$

For both bindings the input places have the tokens needed. However, the guard $[n=k]$ of ReceiveNext evaluates to *false* in the binding RN_1 . Hence, only the binding RN_2 , corresponding to reception of the expected data packet, is enabled in the marking shown in Fig. 2.19. Similarly, we can consider the following two bindings of DiscardPacket:

$$DP_1 = \langle n=1, d="COL", k=2 \rangle$$

$$DP_2 = \langle n=2, d="OUR", k=2 \rangle$$

In this case only the binding DP_1 , corresponding to reception of the data packet that has already been received, is enabled. The reason is that the guard $[n < > k]$ of DiscardPacket evaluates to *false* in the binding DP_2 .

Guards can, in general, be used in many different ways and for many different purposes. Further examples of the use of guards will be given in later chapters.

2.7 Interactive and Automatic Simulation

An execution of a CPN model is described by means of an *occurrence sequence*, which specifies the intermediate markings reached and the steps that occur. A mark-

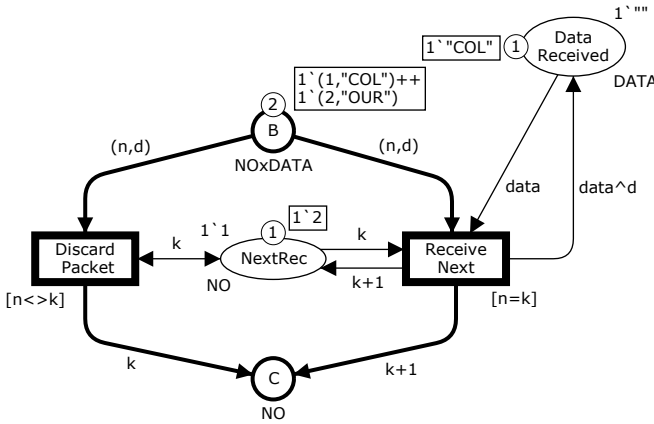


Fig. 2.19 Marking illustrating the semantics of guards

ing that is reachable via an occurrence sequence starting from the initial marking is called a *reachable marking*. The existence of a reachable marking with more than one enabled binding element makes a CPN model *non-deterministic*. This means that there exist different occurrence sequences containing different sequences of steps and leading to different reachable markings. It is important to stress that it is only the choice between the enabled steps which is non-deterministic. The individual steps themselves are deterministic, in the sense that once an enabled step has been selected in a given marking, the marking resulting from its occurrence is uniquely determined, unless a random number function is used in one of the arc expressions.

CPN Tools uses graphical *simulation feedback*, such as that shown in Fig. 2.20, to provide information about the markings that are reached and the binding elements that are enabled and occur during a simulation. The rectangular box next to the transition *ReceivePacket* will be explained shortly.

The tools that are available for simulating CPN models in CPN Tools can be found in the simulation tool palette shown in Fig. 2.21. A VCR (video cassette recorder) metaphor is used for the graphical symbols representing the simulation tools. The simulation tools can be picked up with the mouse cursor and applied to the CPN model. The available tools (from left to right) are:

- Return to the initial marking.
- Stop an ongoing simulation.
- Execute a single transition with a manually chosen binding.
- Execute a single transition with a random binding.
- Execute an occurrence sequence with randomly chosen binding elements interactively (i.e., display the current marking after each step).
- Execute an occurrence sequence with randomly chosen binding elements automatically (i.e., without displaying the current marking after each step).
- Evaluate a CPN ML expression (to be explained in Chap. 3).

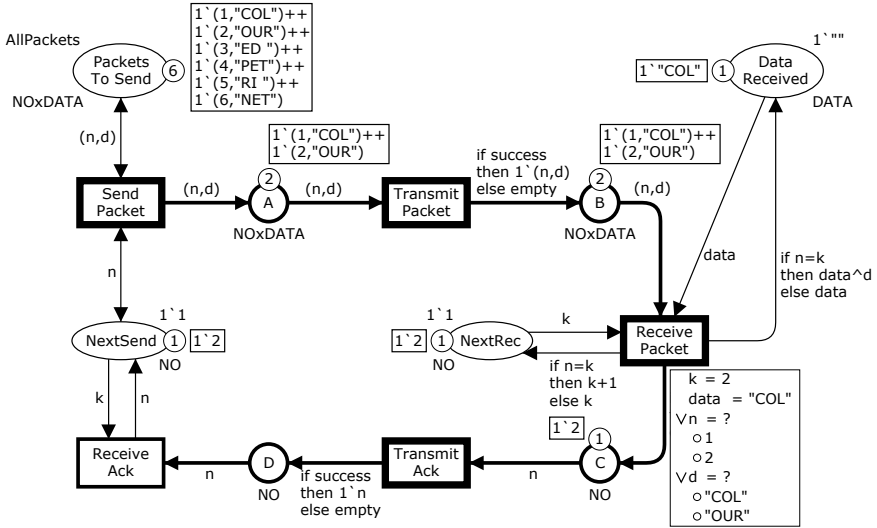


Fig. 2.20 Simulation feedback in CPN Tools



Fig. 2.21 Simulation tool palette in CPN Tools

When a CPN model is simulated in *interactive mode*, the simulator calculates the set of enabled transitions in each marking encountered. It is then up to the user to choose between the enabled transitions and bindings. Figure 2.20 shows an example where the user is in the process of choosing between the enabled binding elements of the transition `ReceivePacket`. The choice between the enabled binding elements is done via the rectangular box opened next to the transition. This box lists the variables of the transition and the values to which they can be bound. In this case, the value 2 has already been bound to the variable `k`, and the value "COL" has been bound to `data`. This is done automatically by the simulator, since there is only one possible choice for these variables. The user still has a choice in binding values to the variables `n` and `d`. The user may also leave the choice to the simulator, which uses a random number generator for this purpose. In the above case it suffices for the user to bind either `n` or `d`, since the value bound to the other variable is then uniquely determined and will be automatically bound by the simulator.

The simulator executes the chosen binding element and presents the new marking and its enabling to the user, who either chooses a new enabled binding element or leaves the choice to the simulator, and so on. This means that it is the simulator that makes all the calculations (of the enabled binding elements and the effect of their occurrences), while it is the user who chooses between the different occurrence se-

quences (i.e., the different behavioural scenarios). An interactive simulation is by its nature slow, since it takes time for the user to investigate the markings and enablings and to choose between them. This means that only a few steps can be executed per minute and the working style is very similar to the single-step debugging known from conventional programming environments.

When a CPN model is simulated in *automatic mode*, the simulator performs all of the calculations and makes all of the choices. This kind of simulation is similar to a program execution, and a speed of several thousand steps per second is typical. Before the start of an automatic simulation, the user specifies one or more stop criteria, for example, that 100 000 transitions shall occur. When one of the stop criteria becomes fulfilled, the simulation stops and the user can inspect the marking which has been reached. There are also a number of different ways in which the user can inspect the markings and the binding elements that occurred during the automatic simulation. We shall briefly return to this at the end of this section.

We have previously illustrated that our CPN model of the protocol possesses non-determinism, concurrency, and conflict. Now let us look at the marking M^* in Fig. 2.22. This marking is one of the many possible results of an automatic simulation. From the marking of *NextRec*, it can be seen that the receiver is expecting data packet number 5, and from the marking of *DataReceived* it can be seen that the receiver has already received the data in the first four data packets in the correct order. However, from the marking of *NextSend*, it follows that the sender is still sending data packet number 4, and a copy of this data packet is present on place B. Since this is not the expected data packet, it will be discarded by the receiver. An acknowledgement requesting data packet number 5 is present at place D. When this is received by the sender, *NextSend* gets the token colour 5, and the sender will start sending data packet number 5.

If the automatic simulation is continued from the marking M^* , we may reach the dead marking M_{dead} shown in Fig. 2.23. Owing to the non-determinism in the CPN model, we cannot guarantee to reach the dead marking since it is possible to keep losing packets and acknowledgements. However, if a dead marking is reached, it will be the marking shown in Fig. 2.23. Here, we see that all six data packets have been received in the correct order. The sender has stopped sending because *NextSend* has the token colour 7 and there is no data packet with the number 7. All of the places A, B, C, and D connecting the network to the sender and receiver are empty. Hence, this marking represents the desired terminal state of the protocol system. By performing a number of automatic simulations of the CPN model starting from the initial marking, it is possible, by means of simulation, to test that the protocol design as captured by the CPN model appears to be correct, in the sense that the protocol succeeds in delivering the data packets in the correct order to the receiver. Conducting a set of automatic simulations does not, however, guarantee that all possible executions of the protocol have been covered. Hence, simulation cannot in general be used to verify properties of the protocol, but it is a powerful technique for testing the protocol and locating errors. In Chap. 7, we introduce state space

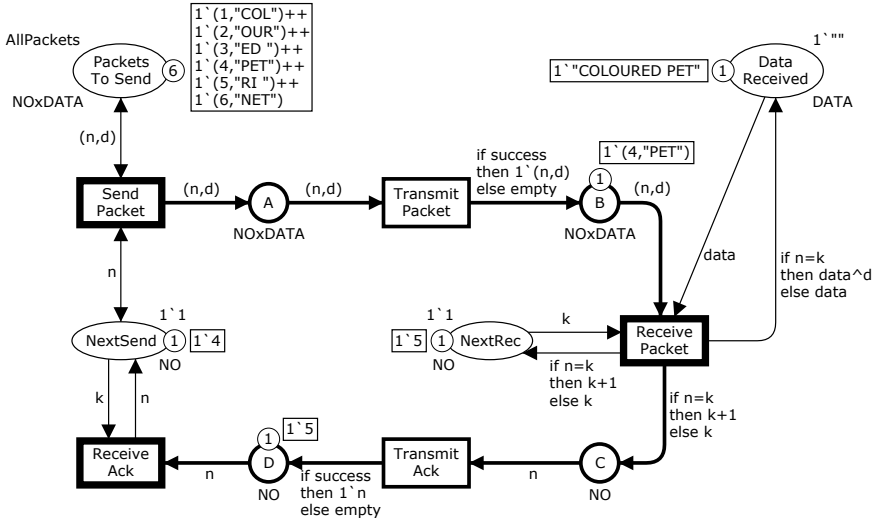


Fig. 2.22 Marking M^* reached by an automatic simulation

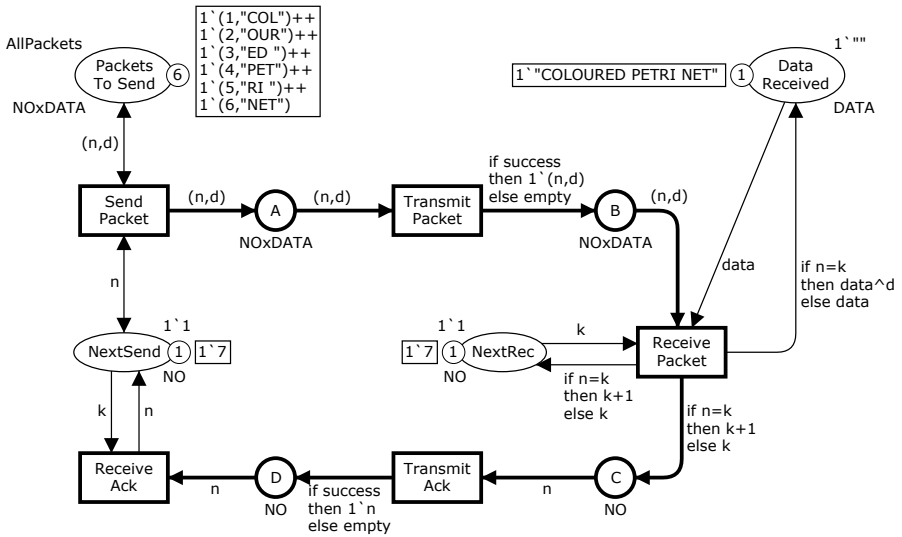


Fig. 2.23 Dead marking M_{dead} reached at the end of an automatic simulation

analysis, which ensures, that all executions are covered. This makes it possible to verify systems, i.e., prove that various behavioural properties are present or absent.

As mentioned earlier in this section, the user may be interested in inspecting some of the markings that were reached and some of the binding elements that occurred during an automatic simulation. A simple (and brute-force) way to do this is to inspect the *simulation report*, which lists the steps that have occurred. For the

simulation described above, the beginning of the simulation report could look as shown in the extract in Fig. 2.24. Here we see the first six transitions that have occurred. The simulation report specifies the name of the occurring transition, the module instance where the transition is located, and the values bound to the variables of the transition. In this case all transitions are in instance 1 of the Protocol module because the CPN model consists of just a single module, named Protocol. The concept of modules in CP-nets will be presented in Chap. 5. The number 0 following the step number specifies the model time at which the transition occurs. Since the model of the protocol presented in this chapter is untimed, all steps occur at time zero. Timed CP-nets will be introduced in Chap. 10.

It is also possible to use graphical visualisation on top of CPN models. These make it possible to observe the execution of the CPN model in a more abstract manner using concepts from the application domain. Figure 2.25 shows an example of a *message sequence chart* (MSC) created from a simulation of the CPN model of the protocol. This MSC has four columns. The leftmost column represents the sender and the rightmost column represents the receiver. The two middle columns represent the sender and receiver sides of the network. The MSC captures a scenario where the first data packet (1, "COL") sent by the sender is lost, as indicated by the small square on the S-Network column. This then causes a retransmission of the data packet. This time, the data packet is successfully transmitted to the receiver and the corresponding acknowledgement 2 is successfully received by the sender.

```

1  0  SendPacket @ (1:Protocol)
   - d = "COL"
   - n = 1
2  0  TransmitPacket @ (1:Protocol)
   - n = 1
   - d = "COL"
   - success = true
3  0  ReceivePacket @ (1:Protocol)
   - k = 1
   - data = ""
   - n = 1
   - d = "COL"
4  0  TransmitAck @ (1:Protocol)
   - n = 2
   - success = true
5  0  ReceiveAck @ (1:Protocol)
   - k = 1
   - n = 2
6  0  SendPacket @ (1:Protocol)
   - d = "OUR"
   - n = 2

```

Fig. 2.24 Beginning of a simulation report

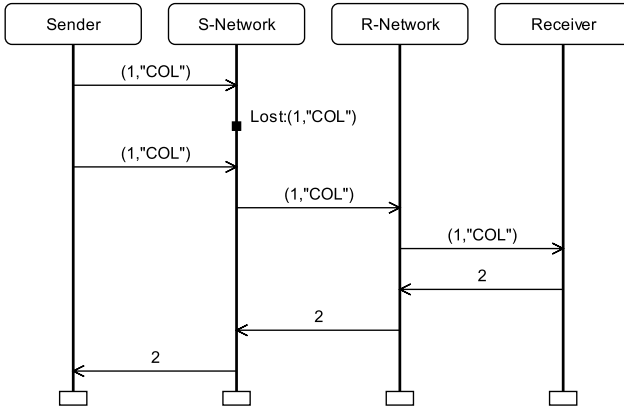


Fig. 2.25 Example of a message sequence chart

In Chap. 13, we give examples of application domain graphics and explain how they can be added to CPN models using the visualisation package [109] provided together with CPN Tools. One of the examples in Chap. 13 also illustrates how graphics can be used to provide input to the CPN model and thereby control its execution.