Chapter 14
Examples of Industrial Applications

This chapter presents a selection of representative projects where CP-nets and their supporting computer tools have been used for system development in an industrial context. These projects have been selected to illustrate the fact that CP-nets can be used in many different phases of system development, ranging from requirements specification to design, validation, and implementation. The CPN models presented were constructed in joint projects between our research group at Aarhus University and industrial partners.

Many CPN projects have been carried out and documented in the literature. Examples of industrial use of CP-nets can be found in the proceedings of the CPN workshops [91], the special issues of the *International Journal on Software Tools for Technology Transfer* [33, 34, 35, 36], and the proceedings of the *International Conferences on Application and Theory of Petri Nets and Other Models of Concurrency* [89]. Many examples have also been published in proceedings and journals related to particular application domains. A comprehensive overview of the applications and industrial use of CP-nets can be found via the Web pages at [40]. The above sources may provide inspiration to people who wish to learn more about the practical application of CP-nets within a particular domain and/or are about to apply CP-nets for the modelling and validation of a larger concurrent system.

Section 14.1 presents a project [69] conducted with Ericsson Telebit, concerned with the design of an edge router discovery protocol for mobile ad hoc networks. Section 14.2 presents a project [64] conducted with Systematic Software Engineering and Aarhus County Hospital, on specifying the business processes at Aarhus County Hospital and identifying requirements for their support by a new IT system. Section 14.3 presents a project [17] conducted with Bang & Olufsen, concerned with the design of the BeoLink system. Finally, Sect. 14.4 presents a project [74] conducted with the Australian Defence Science and Technology Organisation, on the development of a scheduling tool for the Australian Defence Forces. This chapter provides an overview of the CPN modelling and validation conducted in each of these projects. The reader is referred to the papers [17, 64, 69, 73, 74, 75, 112], on which this chapter is based, for further details of these projects.

14.1 Protocol Design at Ericsson Telebit

This project [69] conducted with Ericsson Telebit was concerned with the development of a protocol called the Edge Router Discovery Protocol (ERDP). In the project, a CPN model was constructed that constituted a formal executable specification of ERDP. Simulation and message sequence charts were used in initial investigations of the protocol's behaviour. Then state space analysis was applied to conduct a formal verification of the key properties of ERDP. The modelling, simulation, and subsequent state space analysis all helped to identify several omissions and errors in the design, demonstrating the benefits of using formal techniques in a protocol design process.

14.1.1 Edge Router Discovery Protocol

ERDP is based on the IPv6 protocol suite [56] and supports an edge router in a core network in assigning network address prefixes to gateways in mobile ad hoc networks. A mobile ad hoc network is a collection of mobile nodes, such as laptops, personal digital assistants, and mobile phones, capable of establishing a communication infrastructure for their common use. Ad hoc networks differ from conventional networks in that the nodes in an ad hoc network operate in a fully self-configuring and distributed manner, without any pre-existing communication infrastructure such as designated base stations and routers.

Figure 14.1 shows the network architecture considered in the project. The network architecture consists of an IPv6 stationary core network connecting a number of mobile ad hoc networks on the edge of the core network. A number of *edge routers* reside on the edge of the core network, and each ad hoc network may contain one or more nodes capable of acting as *gateways* for communication with nodes outside the ad hoc network. The edge routers and the gateways handle the connections between the core network and the ad hoc networks, and an edge router may serve multiple ad hoc networks. The core network is a classical wired IP network with stationary nodes, whereas wireless communication is used for communication between the mobile nodes in the ad hoc networks. The edge routers and the gateways are connected via wireless links. The nodes in the individual ad hoc networks may move within an ad hoc network or between ad hoc networks. It is also possible for an entire ad hoc network, including its gateways, to move from one edge router to another edge router, and possibly to be within reach of several edge routers simultaneously.

ERDP is used between the gateways in the ad hoc networks and the edge routers in the core network. ERDP supports gateways in discovering edge routers and supports edge routers in configuring gateways with a globally routeable IPv6 address prefix. This address prefix can then be used to configure global IPv6 unicast addresses for mobile nodes in the ad hoc networks. ERDP is based on an extension

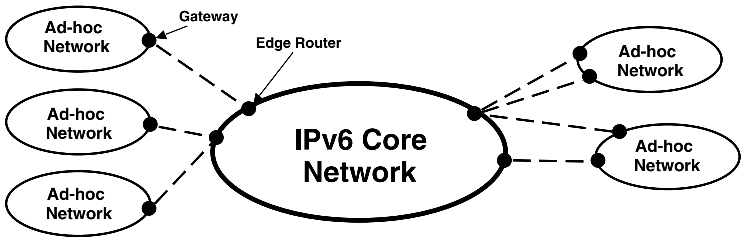


Fig. 14.1 IPv6-based network architecture

of the Neighbor Discovery Protocol (NDP) [85], which is part of the IPv6 protocol suite.

Figure 14.2 shows the basic way that an edge router configures a gateway with an address prefix using ERDP. This message sequence chart (MSC) was generated automatically from the CPN model to be presented in Sect. 14.1.2. The column labelled GWBuffer represents a packet buffer between the gateway protocol entity and the underlying protocol layers. Similarly, the ERBuffer column represents a packet buffer in the edge router. An edge router periodically multicasts unsolicited router advertisements (RAs) to announce its presence to any gateways that may be within reach of that edge router. When an unsolicited RA is received by a gateway, it will reply with its list of currently assigned address prefixes in a unicast router solicitation (RS). In the example shown here, the gateway has no current prefixes and hence it sends an RS with no prefixes (indicated by the empty list [ ]). When the edge router receives the RS, it will consult its lists of available prefixes and in this case select a new address prefix (P1) to be assigned to the gateway. This newly assigned prefix will then be sent back to the gateway in a unicast solicited RA. When the solicited RA containing the prefix is received by the gateway, the gateway will update its lists of currently assigned prefixes to contain the new prefix P1. Prefixes assigned to gateways have a limited lifetime, and hence either will expire or will have to be refreshed by the edge router.

14.1.2 ERDP CPN Model

CP-nets were integrated into the design of ERDP by developing a CPN model of ERDP together with a conventional natural-language specification. The latter is normally used by protocol engineers to specify a protocol, and in the following we refer to the natural-language specification of ERDP as the *ERDP specification*.

Figure 14.3 shows the module hierarchy of the CPN model. The CPN model consists of three main parts. The Gateway module and its four submodules model the operation of the gateway. The EdgeRouter module and its five submodules model the operation of the edge router. The GW_ER_Link module models the wireless communication link between the gateway and the edge router. We have omitted the

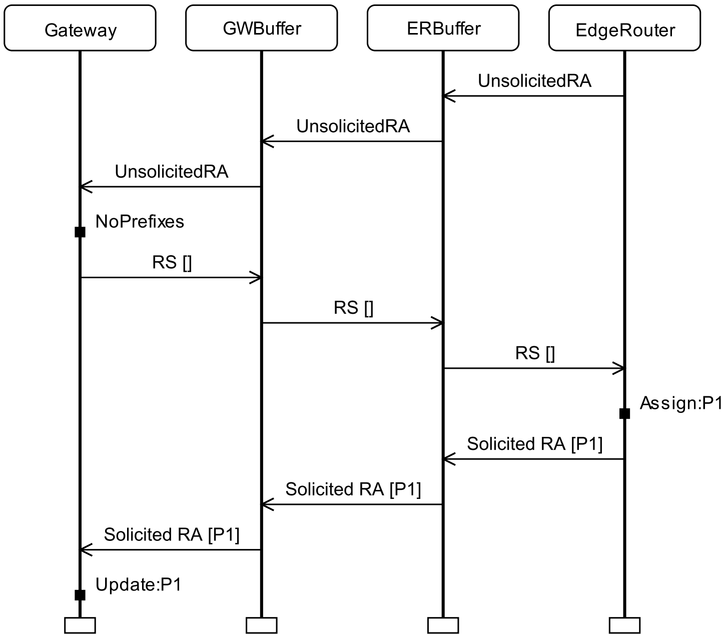


Fig. 14.2 Message sequence chart for prefix configuration with ERDP

names of the substitution transitions on the arcs, since the name of each substitution transition is identical to that of the submodule associated with that substitution transition.

Figure 14.4 shows the ERDP module. The substitution transition Gateway represents the gateway, and the substitution transition EdgeRouter represents the edge router. The communication link between the edge router and the gateway is represented by the substitution transition GW_ER_Link. The four places GWIn, GWOut, ERIn, and EROut model packet buffers between the link layer and the gateway and edge router. Both the gateway (GW) and the edge router (ER) have an incoming and an outgoing packet buffer.

All four places in Fig. 14.4 have the colour set `IPv6Packet`, used to model the IPv6 packets exchanged between the edge routers and gateways. Since ERDP is based on the IPv6 Neighbor Discovery Protocol, the packets are carried as Internet Control Message Protocol (ICMP) packets. The definitions of the colour sets for NDP, ICMP, and IPv6 packets are given in Fig. 14.5 and were derived from RFC 2460 [29], which specifies IPv6 and RFC 2461 [85] specifying NDP. IPv6 addresses and address prefixes are modelled as strings. This makes it possible to use both mnemonic names and standard hexadecimal notation for IPv6 addresses in the CPN model. Protocol fields that do not affect the operation of ERDP have been defined using the colour set `NOTMOD` containing the single dummy value `notmod`. These

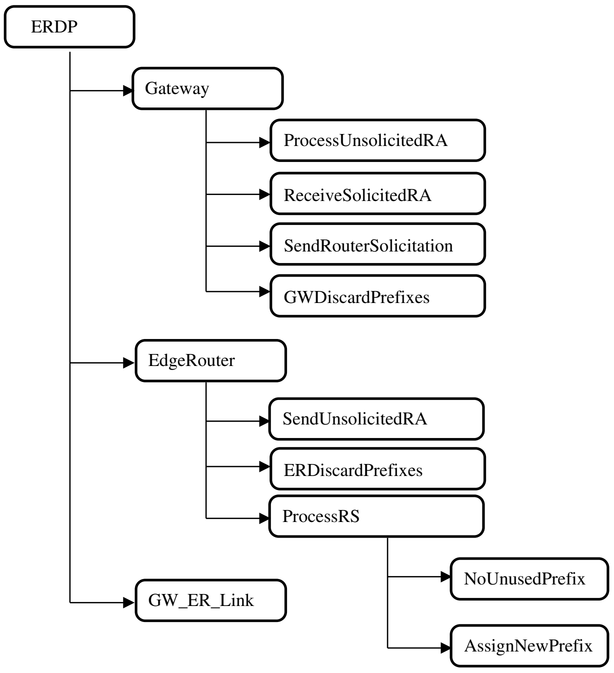


Fig. 14.3 Module hierarchy of the ERDP model

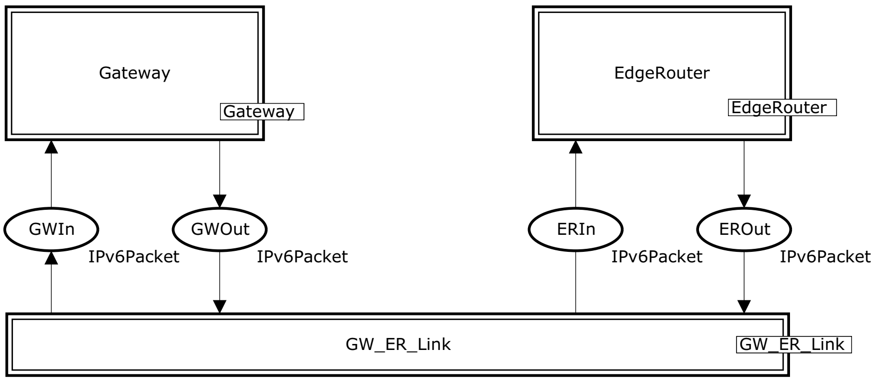


Fig. 14.4 ERDP module

fields could alternatively have been omitted, but it was considered important for later implementations of ERDP that the tokens in the CPN model should have the same set of fields as the packets in the implementation. The colour sets `UInt32`, `UInt16`, `UInt8`, `Bit8`, and `Bit4` are all defined as `INT`. They model bit fields in the packets and are defined as integers, as we are not concerned with the specific bit

```
(* --- IPv6 addresses --- *)
colset IPv6Addr = string;

(* --- Router Solicitations --- *)
colset RSOption = union
                    RS_SrcLinkAddr       : NDLinkAddrOption +
                    RS_PrefixInformation : NDPrefixInfoOption;

colset RSOptions = list RSOption;

colset RouterSolicitation = record Options : RSOptions *
                                   NU      : NOTMOD;

(* --- Router Advertisements --- *)
colset RAOption = union
                    RA_SrcLinkAddr       : NDLinkAddrOption +
                    RA_MTU               : NDMTUOption      +
                    RA_PrefixInformation : NDPrefixInfoOption;

colset RAOptions = list RAOption;

colset RouterAdvertisement = record CurHopLimit    : UInt8  *
                                    M              : Bit    *
                                    O              : Bit    *
                                    RouterLifetime : UInt16 *
                                    ReachableTime  : UInt32 *
                                    RetransTimer   : UInt32 *
                                    Options        : RAOptions;

(* --- ICMP messages --- *)
colset ICMPBody = union RS : RouterSolicitation +
                        RA : RouterAdvertisement;

colset ICMPMessage = record Type : UInt8 *
                            Code : UInt8 *
                            Message : ICMPBody;

(* --- IPv6 packets --- *)
colset IPv6Payload = union ICMP : ICMPMessage;

colset IPv6Header = record Version       : Bit4     *
                           TrafficClass  : NOTMOD   *
                           Flowlabel     : NOTMOD   *
                           PayloadLength : NOTMOD   *
                           NextHeader    : Bit8     *
                           HopLimit      : Bit8     *
                           SourceAddress : IPv6Addr *
                           DestAddress   : IPv6Addr;

colset IPv6Packet = record Header     : IPv6Header *
                           ExtHeaders : NOTMOD *
                           Payload    : IPv6Payload;
```

Fig. 14.5 Declarations for IPv6 and ICMP packets

layout of packets, but only the semantics of the individual packet fields. The colour set `Bit` is defined as `BOOL`.

Figure 14.6 shows the EdgeRouter module. The places ERIn and EROut are related to the accordingly named socket places in the ERDP module (see Fig. 14.4). The place Config models the configuration information associated with the edge router, and the place PrefixCount models the number of prefixes still available in the edge router for distribution to gateways. The place PrefixAssigned is used to keep track of which prefixes are assigned to which gateways.

Figure 14.7 shows the declarations of the colour sets for the three places in Fig. 14.6. The configuration information for the edge router (modelled by the colour set `ERConfig`) is a record consisting of the IPv6 link-local address and the link-layer address of the edge router. A list of pairs (colour set `ERPrefixAssigned`) consisting of a link-local address and a prefix is used to keep track of which prefixes are assigned to which gateways. A counter modelled by the place PrefixCount with the colour set `PrefixCount` is used to keep track of the number of prefixes still available. When this counter reaches 0, the edge router has no further prefixes available for distribution. The number of available prefixes can be modified by changing the initial marking of the place PrefixCount, which is set to 1 by default.

The substitution transition SendUnsolicitedRA (in Fig. 14.6) corresponds to the multicasting of periodic unsolicited RAs by the edge router. The substitution transition ProcessRS models the reception of unicast RSs from gateways, and the sending of a unicast RA in response. The substitution transition ERDiscardPrefixes models the expiration of prefixes on the edge router side.

The marking shown in Fig. 14.6 has a single token on each of the three places used to model the internal state of the edge router protocol entity. In the marking shown, the token on the place PrefixAssigned with the colour `[]` corresponds to the edge router not having assigned any prefixes to the gateways. The token on the place

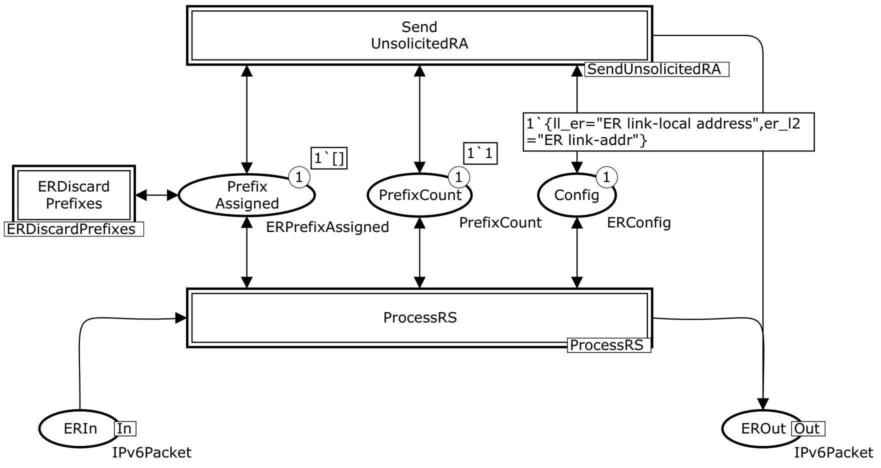


Fig. 14.6 EdgeRouter module

```
colset LinkAddr    = string;

colset ERConfig = record
                   ll_er : IPv6Addr * (* link-local address  *)
                   er_l2 : LinkAddr;  (* link-addr (layer 2) *)

colset ERPrefixEntry    = product IPv6Addr * IPv6Prefix;
colset ERPrefixAssigned = list ERPrefixEntry;

colset PrefixCount = int;
```

Fig. 14.7 Colour set definitions for edge routers

PrefixCount with colour `1` indicates that the edge router has a single prefix available for distribution. Finally, the colour of the token on the place Config specifies the link-local and link addresses of the edge router. In this case the edge router has the symbolic link-local address `ER link-local address`, and the symbolic link-address `ER link-addr`.

Figure 14.8 depicts the SendUnsolicitedRA module which is the submodule of the substitution transition SendUnsolicitedRA in Fig. 14.6. The transition SendUnsolicitedRA models the sending of the periodic unsolicited router advertisements. The variable `erconfig` is of type `ERConfig`, and the variable `prefixleft` is of type `PrefixCount` (see Fig. 14.7). The transition SendUnsolicitedRA is enabled only if the edge router has prefixes available for distribution, i.e., `prefixleft` is greater than 0. This is ensured by the function `SendUnsolicitedRA` in the guard of the transition.

Figure 14.9 depicts the marking of the SendUnsolicitedRA module after the occurrence of the transition SendUnsolicitedRA in the marking shown in Fig. 14.8. An unsolicited router advertisement has been put in the outgoing buffer of the

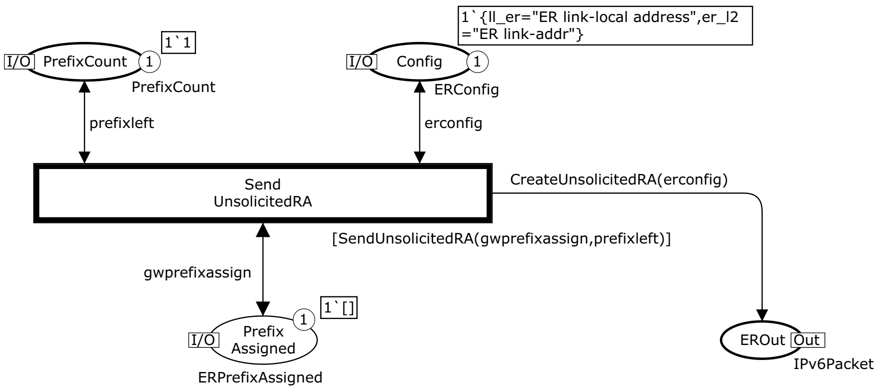
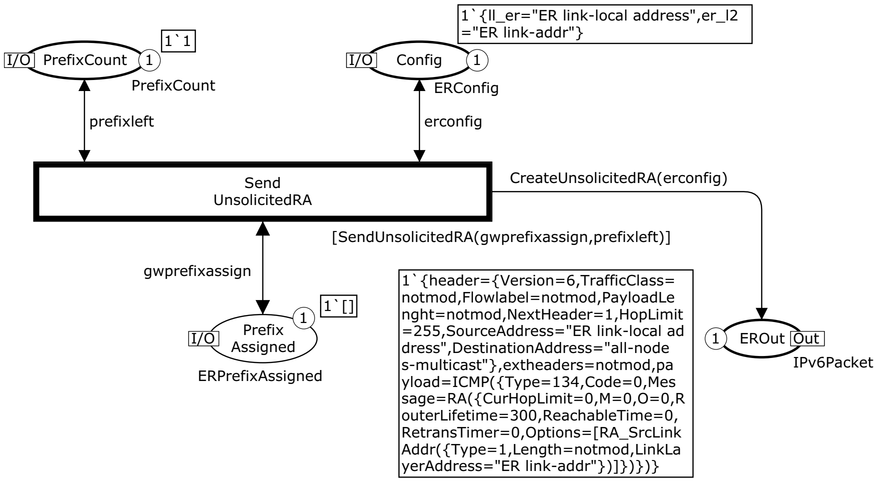


Fig. 14.8 Initial marking of the SendUnsolicitedRA module



I/O PrefixCount 1 1`1
PrefixCount
I/O Config 1
1`{ll_er="ER link-local address",er_l2="ER link-addr"}
ERConfig
prefixleft
erconfig
Send UnsolicitedRA
CreateUnsolicitedRA(erconfig)
[SendUnsolicitedRA(gwprefixassign,prefixleft)]
gwprefixassign
I/O Prefix Assigned 1 1`[]
ERPrefixAssigned
1`{header={Version=6,TrafficClass=notmod,Flowlabel=notmod,PayloadLenght=notmod,NextHeader=1,HopLimit=255,SourceAddress="ER link-local address",DestinationAddress="all-nodes-multicast"},extheaders=notmod,payload=ICMP({Type=134,Code=0,Message=RA({CurHopLimit=0,M=0,O=0,RouterLifetime=300,ReachableTime=0,RetransTimer=0,Options=[RA_SrcLinkAddr({Type=1,Length=notmod,LinkLayerAddress="ER link-addr"})]})})}
1 EROut Out
IPv6Packet

Fig. 14.9 Module SendUnsolicitedRA, after occurrence of SendUnsolicitedRA

edge router. It can be seen that the `DestinationAddress` is the address `all-nodes-multicast`, the `SourceAddress` is `ER link-local address`, and the `LinkLayerAddress` (in the options part) is `ER link-addr`.

Figure 14.10 shows the part of the GW_ER_Link module that models transmission of packets from the edge router to the gateway across the wireless link. Transmission of packets from the gateway to the edge router is modelled similarly. The places GWIn and EROut are linked to the similarly named socket places in Fig. 14.4. The transition ERtoGW models the successful transmission of packets, whereas the transition LossERtoGW models the loss of packets. The variable `ipv6packet` is of type `IPv6Packet`. A successful transmission of a packet from the edge router to the gateway corresponds to moving the token modelling the packet from the place EROut to GWIn. If the packet is lost, the token will only be removed from the place EROut.

Wireless links, in general, have a lower bandwidth and higher error rate than wired links. These characteristics have been abstracted away in the CPN model since our aim is to reason not about the performance of ERDP but rather its logical

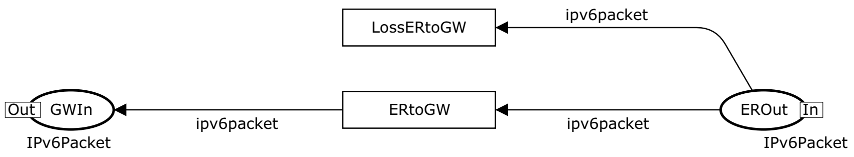


Fig. 14.10 Part of the GW_ER_Link module

correctness. Duplication and reordering of messages are not possible on typical one-hop wireless links, since the detection of duplicates and the preservation of order are handled by the data-link layer. The modelling of the wireless links does allow overtaking of packets, but this overtaking is eliminated in the analysis phase described in Sect. 14.1.3 where we impose bounds on the capacity of the input and output packet buffers.

The CPN model was developed as an integrated part of the development of ERDP. The creation of the CPN model was done in cooperation with the protocol engineers at Ericsson Telebit and in parallel with the development of the ERDP specification. Altogether, 70 person-hours were spent on CPN modelling. The protocol developers at Ericsson Telebit were given a 6 hour course on the CPN modelling language. This course enabled them to read and interpret CPN models, allowing the CPN model to be used as a basis for discussions of the protocol design and its representation as a CPN model.

The development of ERDP started out with the creation of an initial natural-language specification. Based on this specification, an initial version of the CPN model was created. The act of creating this initial CPN model and discussing it, in Review 1, led to the identification of several issues related to the design and operation of ERDP. This included design errors, incompleteness and ambiguities in the specification, and ideas for simplifications and improvements of the protocol design. Based on the issues discovered in Review 1, the ERDP specification was revised and extended. The CPN model was then revised, and a second review, was performed. Review 2 led to further identification of issues, which were eventually resolved, and the ERDP specification was modified accordingly. The CPN model was then modified again to reflect the revised ERDP specification. At this stage, no further issues were discovered in the process of revising the CPN model.

Table 14.1 categorises and enumerates the issues encountered in each of the two reviews. These issues were identified in the process of constructing the CPN model, performing single-step executions of the CPN model, and conducting discussions of the CPN model among the project group members. Altogether, 24 issues were identified.

Message sequence charts (such as the one shown in Fig. 14.2), integrated with simulation were used in both review steps to investigate the behaviour of ERDP in

Table 14.1 Issues encountered in the modelling phase

Category	Review 1	Review 2	Total
Errors in protocol specification/operation	2	7	9 issues
Incompleteness and ambiguity in specification	3	6	9 issues
Simplifications of protocol operation	2	0	2 issues
Additions to the protocol operation	4	0	4 issues
Total	11	13	24 issues

detail. The use of MSCs in the project was of particular relevance since it presented the operation of the protocol in a form well known to protocol developers.

The construction of a CPN model can be seen as a very thorough and systematic way of reviewing a design specification of a protocol. Using an iterative process where both a conventional natural-language specification and a CPN model were developed (as in this project) turned out to be an effective way of integrating CPN modelling and analysis into the development of a protocol. In general, we believe that a combination of an executable formal model (such as a CPN model) and a natural-language specification provides a useful way to develop a protocol. One reason why both are required is that the people who are going to implement the protocol are unlikely to be familiar with CP-nets. Secondly, in the present case, there are important parts of the ERDP specification that are not reflected in the CPN model, such as the layout of packets.

14.1.3 State Space Analysis and Verification

State space analysis was pursued after the three iterations of modelling described in the previous subsection. The purpose of the state space analysis was to conduct a more thorough investigation of the operation of ERDP, including verification of its key properties.

The first step towards state space analysis of the CPN model was to obtain a finite state space. The CPN model presented in the previous subsection has an infinite state space, since an arbitrary number of tokens (packets) can be put on the places modelling the packet buffers. As an example, the edge router may initially send an arbitrary number of unsolicited router advertisements. To obtain a finite state space, an upper integer bound of 1 was imposed on each of the places GWIn, GWOut, ERIn, and EROut (see Fig. 14.4) that model the packet buffers. This also prevents overtaking among the packets transmitted across the wireless link. Furthermore, the number of packets simultaneously present in the four input/output buffers was limited to 2. Technically, this was done by using the *branching options* available in the CPN state space tool to prevent the processing of enabled transitions whose occurrence in a given marking would violate the above bounds.

First, we generated the state space for the considered configuration of the protocol. This was followed by generation of the state space report and the use of user-defined queries to investigate the model-dependent properties of the protocol. The key property of ERDP is proper configuration of the gateway with prefixes. This means that for a given prefix and state where the gateway has not yet been configured with that prefix, the protocol must be able to configure the gateway with that prefix. Furthermore, when the gateway has been configured with the prefix, the edge router and the gateway should be *consistently configured*, i.e., the assignment of the prefix must be recorded both in the gateway protocol entity and in the edge router protocol entity. Whether a marking represents a consistently configured state for a given prefix can be checked by inspecting the marking of the place PrefixAssigned in the edge router and the marking of the place Prefixes in the gateway.

The state space analysis was conducted in three steps. The first step was to consider the simplest possible configurations of ERDP, starting with a single prefix and assuming that there is no loss of packets on the wireless link and that prefixes do not expire. The full state space for this configuration had 46 nodes and 65 arcs. The SCC graph had 36 nodes and 48 arcs. Inspection of the state space report showed that there was a single dead marking represented by node 36. Inspection of this node showed that it represented a state where all of the packet buffers were empty, but where the edge router and gateway were inconsistently configured in the sense that the edge router had assigned the prefix P1 (the single prefix), while the gateway was not configured with that prefix. This was an error in the protocol. To locate the source of the problem, query functions in the state space tool were used to obtain a counterexample leading from the node representing the initial marking to node 36. Figure 14.11 shows the resulting error trace, visualised by means of an MSC. The problem is that the edge router sends two unsolicited RAs. The first one gets through and the gateway is configured with the prefix, which can be seen from the event marked with *A* in the lower part of the MSC. However, when the second RS, without any prefixes, is received by the edge router (the event marked with *B*), the corresponding solicited RA will not contain any prefixes. Because of the way the protocol was specified, the gateway will therefore update its list of prefixes to the empty list (the event marked with *C*), and the gateway is no longer configured with a prefix.

To fix the error, the protocol was modified such that the edge router always replied with the list of all prefixes that it had currently assigned to the gateway. The state space for the modified protocol consisted of 34 nodes and 49 arcs, and there were no dead markings in the state space. The state space report specified that there were 11 home markings (represented by the nodes in the single terminal SCC). Inspection of these 11 markings showed that they all represented consistently configured states for the prefix P1. The markings were contained in the single terminal SCC of the state space. This shows that, from the initial marking it is always possible to reach a consistently configured state for the prefix, and that when such a marking has been reached, the protocol entities will remain in a consistently configured state. To verify that a consistently configured state would eventually be reached, it was checked that the single terminal SCC was the only non-trivial SCC. This showed that all cycles in the state space (which correspond to non-terminating executions of the protocol) were contained in the terminal SCC, which (from above) contained only consistently configured states. The reason why the protocol is not supposed to terminate in a consistently configured state represented by a dead marking is that the gateway may, at any time, when it is configured, send a router solicitation back to the edge router to have its prefixes refreshed. Since we are ignoring expiration of prefixes, the edge router will always refresh the prefix.

When the correctness of the protocol had been established for a single prefix, we increased the number of prefixes. When there is more than one prefix available it no longer holds that a marking will *eventually* be reached where *all* prefixes are consis-

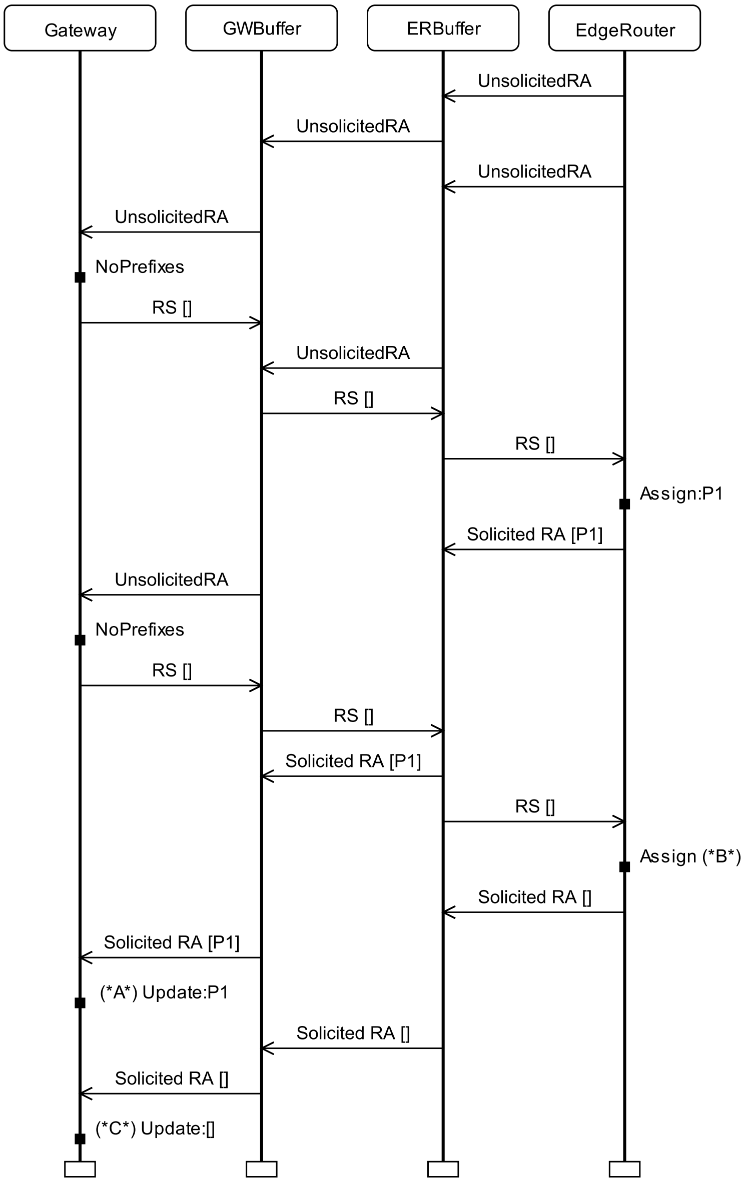


Fig. 14.11 Message sequence chart showing an execution leading to an undesired terminal state

tently configured. The reason is that with more than one prefix, the edge router may at any time decide not to configure the gateway with additional prefixes. Hence, a state where all prefixes have been consistently configured might not eventually be reached. Instead, firstly, it was verified that there was a single terminal SCC, all markings of which represent states where all prefixes have been consistently configured. This shows that it is always possible to reach such a marking, and when the protocol has consistently configured all prefixes, the protocol entities will remain consistently configured. Secondly, it was checked that all markings in each non-trivial SCC represented markings where the protocol entities were consistently configured with a subset of the prefixes available in the edge router.

The second step was to allow packet loss on the wireless link between the edge router and the gateway. First, the case was considered in which there is only a single prefix for distribution. The state space for this configuration had 40 nodes and 81 arcs. Inspection of the state space report showed that there was a single dead marking. This marking represented an undesired terminal state where the prefix had been assigned by the edge router, but the gateway was not configured with the prefix. Figure 14.12 shows an MSC corresponding to a path in the state space from the initial marking to the undesired dead marking. The problem is that when the solicited RA containing the prefix is lost, the edge router will have assigned its last prefix and is no longer sending any unsolicited RAs. Furthermore, there are no timeouts in the protocol entities that could trigger a retransmission of the prefix to the gateway.

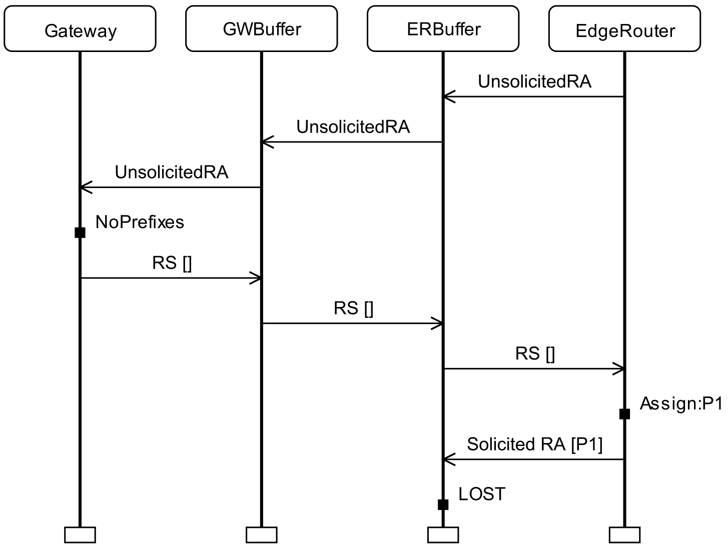


Fig. 14.12 Message sequence chart showing an execution leading to an undesired terminal state

The problem identified above was fixed by ensuring that the edge router would resend an unsolicited RA to the gateway as long as it had prefixes assigned to the gateway. The state space of the revised CPN model had 68 nodes and 160 arcs. Inspection of the state space report showed that there were no dead markings and no home markings. Investigation of the terminal SCCs showed that there were two terminal SCCs, each containing 20 markings. The nodes in one of them all represented states where the edge router and gateway were consistently configured with the single prefix P1, whereas the nodes in the other terminal SCC all represented states where the protocol entities were not consistently configured. The markings in the undesired terminal SCC represent a livelock in the protocol, i.e., if one of the markings in the undesired terminal SCC is reached, it is no longer possible to reach a state where the protocol entities are consistently configured with the prefix. The source of the livelock was related to the control fields used in the router advertisements for refreshing prefixes and their interpretation in the gateway. This was identified by obtaining the MSC for a path leading from the initial marking to one of the markings in the undesired terminal SCC. As a result, the processing of router advertisements in the gateway was modified. The state space for the protocol with the modified processing of router advertisements also had 68 nodes and 160 arcs. The state space had a single terminal SCC containing 20 nodes, which all represented states where the protocol entities were consistently configured with the single prefix.

When packet loss is present, it is not immediately possible to prove that the two protocol entities will eventually be consistently configured. The reason is that any number of packets can be lost on the wireless link. Each of the non-trivial SCCs was inspected using a user-defined query to investigate the circumstances under which the protocol entities would not eventually be consistently configured. This query checked that either all nodes in the non-trivial SCC represented consistently configured states or none of the nodes in the SCC represented a consistently configured state. For those non-trivial SCCs where no node represented a consistently configured state, it was checked that all cycles contained the occurrence of a transition corresponding to loss of a packet. Since this was the case, it can be concluded that any failure to reach a consistently configured states will be due to packet loss and nothing else. Hence, if only finitely many packets are lost, a consistently configured state for some prefix will *eventually* be reached.

The third and final step in the analysis was to allow prefixes to expire. The analysis was conducted first for a configuration where the edge router had only a single prefix to distribute. The state space for this configuration had 173 nodes and 531 arcs. The state space had a single dead marking, and inspection of this dead marking showed that it represented a state where the edge router has no further prefixes to distribute, it has no prefixes recorded for the gateway, and the gateway is not configured with any prefix. This marking is a desired terminating state of the protocol, as we expect prefixes to eventually expire. Since the edge router has only finitely many prefixes to distribute, the protocol should eventually terminate in such a state. The single dead marking was also a home marking, meaning that the protocol can always enter the expected terminal state.

When prefixes can expire, it is possible that the two protocol entities may never enter a consistently configured state. The reason is that a prefix may expire in the edge router (although this is unlikely) before the gateway has been successfully configured with that prefix. Hence, we are only able to prove that for any marking where a prefix is still available in the edge router, it is possible to reach a marking where the gateway and the edge router are consistently configured with that prefix.

Table 14.2 lists statistics for the size of the state space in the three verification steps for different numbers of prefixes. The column '|P|' specifies the number of prefixes. The columns 'Nodes' and 'Arcs' give the numbers of nodes and arcs, respectively, in the state space. For the state spaces obtained in the first verification step, it can be seen that 38 markings and 72 arcs are added for each additional prefix. The reason for this is that ERDP proceeds in phases where the edge router assigns prefixes to the gateway one at a time. Configuring the gateway with an additional prefix follows exactly the same procedure as that for the assignment of the first prefix. Once the state space had been generated, the verification of properties could be done in a few seconds. It is also worth observing that as the assumptions are relaxed, i.e., we move from one verification step to the next, the sizes of the state spaces grow. This, combined with the identification of errors in the protocol even in the simplest configuration, without packet loss and without expiration of prefixes, shows the benefit of starting state space analysis from the simplest configuration and then gradually lifting the assumptions. Furthermore, the state explosion problem was not encountered during the verification of ERDP, and the key properties of ERDP were verified for the number of prefixes that were envisioned to appear in practice.

It can be argued whether or not the issues and errors discovered in the process of modelling and conducting state space analysis would have been identified if additional conventional reviews of the ERDP specification had been conducted. Some of them probably would have been, but more subtle problems such as the inconsis-

Table 14.2 State space statistics for the three verification steps

	No loss/No expire		Loss/No Expire		Loss/Expire	
|P|	Nodes	Arcs	Nodes	Arcs	Nodes	Arcs
1	34	49	68	160	173	531
2	72	121	172	425	714	2 404
3	110	193	337	851	2 147	7 562
4	148	265	582	1 489	5 390	19 516
5	186	337	926	2 390	11 907	43 976
6	224	409	1 388	3 605	23 905	89 654
7	262	481	1 987	5 185	44 550	169 169
8	300	553	2 742	7 181	78 211	300 072
9	338	625	3 672	9 644	130 732	505 992
10	376	697	4 796	12 625	209 732	817 903

tent configurations discovered during state space analysis would probably not have been discovered until the first implementation of ERDP was operational. The reason for this is that discovering these problems requires one to consider subtle execution sequences of the protocol, and there are too many of these to do this in a systematic way. This demonstrates the value of being able to conduct state space analysis of a CPN model and in this way cover all execution sequences.

14.1.4 Conclusions from the ERDP Project

This project showed that even the act of constructing a CPN model based on the ERDP specification provided valuable input to the ERDP specification, and the use of simulation added further insight into the operation of the protocol. State space analysis, starting with the simplest possible configuration of the protocol, identified additional errors in the protocol. The state space analysis succeeded in establishing the key properties of ERDP.

Overall, the application of CP-nets in the development of ERDP was considered a success for three main reasons. Firstly, it was demonstrated that the CPN modelling language and supporting computer tools were powerful enough to specify and analyse a real-world communication protocol and that they could be integrated into the conventional protocol development process. Secondly, the act of constructing the CPN model, executing it, and discussing it led to the identification of several non-trivial design errors and issues that, under normal circumstances, would not have been discovered until, at best, the implementation phase. Finally, the effort of constructing the CPN model and conducting the state space analysis was represented by approximately 100 person-hours. This is a relatively small investment compared with the many issues that were identified and resolved early as a consequence of constructing and analysing the CPN model.

14.2 Requirements Engineering at Systematic

This project [64, 75], conducted with Systematic Software Engineering and Aarhus County Hospital was concerned with specifying the business processes at Aarhus County Hospital and their support by a new IT System, called the Pervasive Health Care System (PHCS). A CPN model of PHCS was used to engineer requirements for the system, and input from nurses was crucial in this process. The project demonstrated how behavioural visualisation driven by a CPN model can be used to visualise system behaviour and enable the engineering of requirements through discussions with people who are not familiar with the CPN modelling language.

14.2.1 Pervasive Health Care System

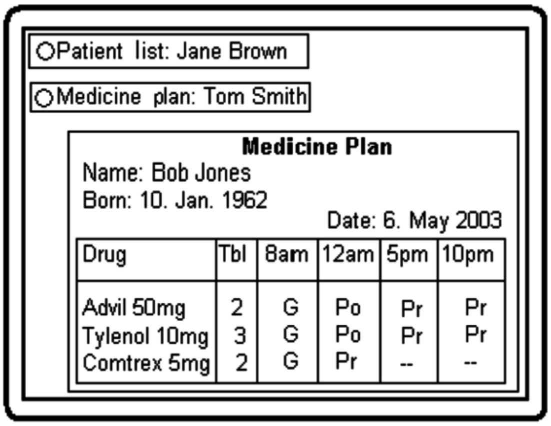
The aim of PHCS is to improve the system for electronic patient records (EPR) deployed at the hospitals in Aarhus, Denmark. EPR is a comprehensive health care IT system with a budget of approximately 15 million US dollars; it will eventually have 8–10,000 users.

EPR solves obvious problems that occur with paper-based patient records such as being not always up-to-date, only present in one location at a time, misplaced, or sometimes even lost. However, the version of EPR that was deployed at the time of the project was a desktop-PC-based system, which is not very practical for hospital work, since users such as nurses and doctors are often on the move and away from their offices (and thus their desktop PCs). Moreover, users are frequently interrupted. Therefore, the desktop-PC-based EPR potentially induces at least two central problems for its users. The first problem is *immobility*: in contrast to a paper-based record, an electronic patient record accessed only from desktop PCs cannot be easily transported. The second problem is *time-consuming login and navigation*: EPR requires user identification and login to ensure the confidentiality and integrity of information, and to start using the system for clinical work, a logged-in user must navigate to find a specific document for a given patient, for example.

The motivation for PHCS is to address these problems. In the ideal situation, the users should have access to the IT system wherever they need it, and it should be easy to resume a work process which has been interrupted. The use of personal digital assistants (PDAs), with which nurses and doctors could access EPR using a wireless network, is a possible solution to the immobility problem. That approach has been considered, but it is not ideal, for example, because of well-known characteristics of PDAs such as small screens and limited memory, and because it does not fully address the time-consuming login and navigation problem.

PHCS is a more ambitious solution, which takes advantage of the possibilities of pervasive computing to a greater extent. Three basic design principles are exploited. The first principle is that PHCS is *context-aware*: nurses, patients, beds, medicine trays, and other items are equipped with radio frequency identity (RFID) tags, enabling the presence of such items to be detected automatically, for example, by computers located beside the medicine cabinet and the patient beds. The second design principle is that PHCS is *propositional*, in the sense that it makes qualified propositions, or guesses. Context changes may result in the automatic generation of buttons that appear on the taskbars of computers. Users may explicitly accept a proposition by clicking on a button, or implicitly ignore or reject it by not clicking. As an example, the presence of a nurse holding a medicine tray for patient P in front of the medicine cabinet is a context that triggers the automatic generation of a button Medicine plan:P on the computer in the medicine room. If the nurse clicks the button, he/she is logged in and taken to P's medicine plan. The third design principle is that PHCS is *non-intrusive*, i.e., it does not interfere with or interrupt hospital work processes in an undesired way. Thus, when a nurse approaches a computer, it should react to his/her presence in such a way that a second nurse, who may currently be working on the computer, is not disturbed or interrupted.

Figure 14.13 presents a simplified interface of PHCS. The current context of the system is that nurse Jane Brown is engaged in pouring medicine for patient Bob Jones, to be given at 12 a.m. The medicine plan on the display shows which medicines have been prescribed (indicated by Pr), poured (Po), and given (G) at the current time. It can be seen that Advil and Tylenol have been poured for 12 a.m., but Comtrex has not yet peen poured. Moreover, the medicine tray for another patient, Tom Smith, stands close to the computer, as can be seen from the taskbar buttons.



Medicine Plan

Name: Bob Jones
Born: 10. Jan. 1962
Date: 6. May 2003

Drug	Tbl	8am	12am	5pm	10pm
Advil 50mg	2	G	Po	Pr	Pr
Tylenol 10mg	3	G	Po	Pr	Pr
Comtrex 5mg	2	G	Pr	--	--

Fig. 14.13 Outline of simplified PHCS interface

14.2.2 PHCS CPN Model

The CPN models of the envisioned new work processes and of the proposed computer support were created with a focus on the *medicine administration* work process. Assume that nurse N wants to pour medicine into a medicine tray and give it to patient P. First, N goes to the room containing the medicine cabinet (the medicine room). Here, there is a context-aware computer on which the buttons Login:N and Patient list:N appear on the taskbar when N approaches. If the second button is clicked, N is logged in and a list of the patients whom N is in charge of is displayed on the computer. A medicine tray is associated with each patient. When N takes P's tray near the computer, the button Medicine plan:P will appear on the taskbar, and a click will make P's medicine plan appear on the display. N pours the prescribed medicine into the tray and acknowledges this in PHCS. When N leaves the medicine room, he/she is automatically logged out. N now takes P's medicine tray and goes to the ward where P lies in a bed, which is supplied with a context-aware computer. When N approaches, the buttons Login:N, Patient list:N, and Medicine plan:P appear on the taskbar. If the last button is clicked, the medicine plan for P is displayed. Finally, N gives the medicine tray to P and acknowledges this in PHCS. When N leaves the bed area, he/she is automatically logged out.

The description given above captures just one specific combination of work processes. There are numerous other scenarios to take into account: for example, medicine may be poured for one or more patients, for only one round of medicine giving, for all four regular rounds of a 24 hour period, or for ad hoc giving; a nurse may have to fetch trays left in the wards prior to pouring; a nurse may approach the medicine cabinet without intending to pour medicine, but instead only to log into EPR (via PHCS) or to check an already filled medicine tray; or two or more nurses may do medicine administration at the same time. To support a smooth medicine administration work process, the requirements for PHCS must deal with all of these scenarios and many more. A CPN model, with its fine-grained and coherent nature, is able to support the investigation and validation of this.

Figure 14.14 shows the module hierarchy of the medicine administration CPN model. The organisation of the modules reflects how the work process of medicine administration is decomposed into smaller work processes. We can give an impression of the model by describing the module shown in Fig. 14.15. This module models the pouring and checking of trays and is represented by the node PourCheckTrays in Fig. 14.14. The medicine cabinet computer is in focus. It is modelled by a token on the place MedicineCabinetComputer. This place has the colour set `COMPUTER`, whose elements are 4-tuples `(compid,display,taskbar,users)` consisting of a computer identification, its display (main screen), its taskbar buttons, and its current users. In the initial marking, the computer has a blank display, no taskbar buttons, and no users.

The colour set `NURSE` is used to model nurses. A nurse is represented as a pair `(nurse,trays)`, where `nurse` identifies the nurse and `trays` is a list holding

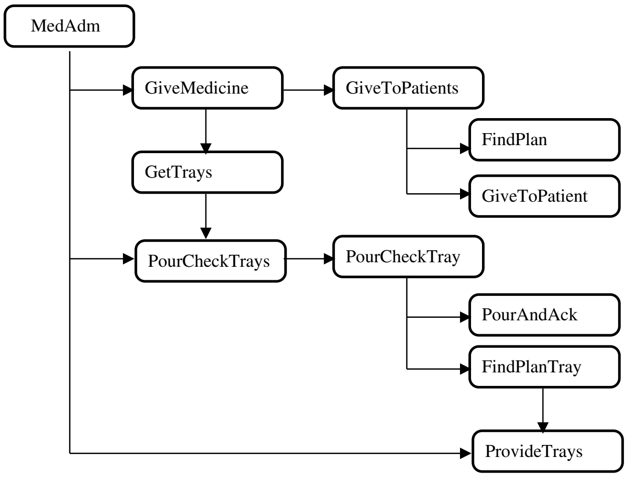


Fig. 14.14 Module hierarchy of the PHCS medicine administration model

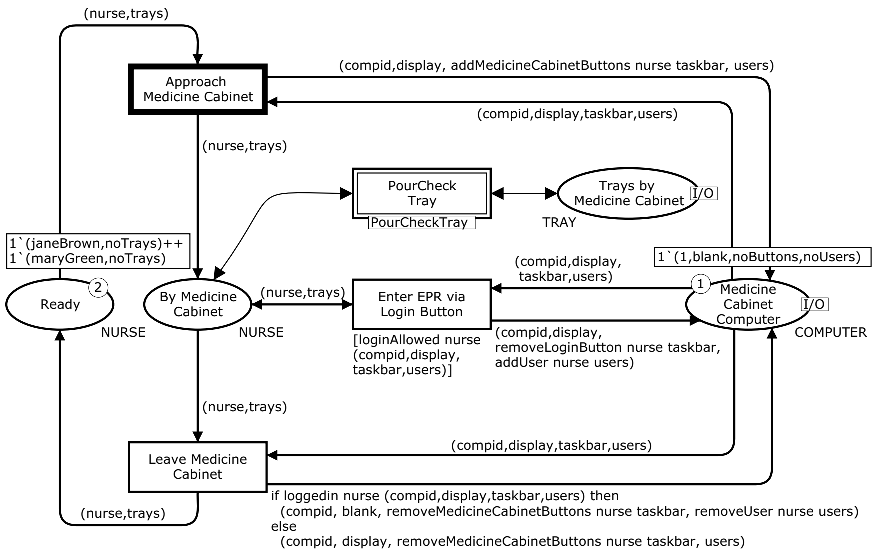

(nurse,trays)
Approach Medicine Cabinet
(compid,display, addMedicineCabinetButtons nurse taskbar, users)
(compid,display,taskbar,users)
(nurse,trays)
PourCheck Tray
PourCheckTray
Trays by Medicine Cabinet
I/O
TRAY
1`(janeBrown,noTrays)++
1`(maryGreen,noTrays)
(compid,display, taskbar,users)
1`(1,blank,noButtons,noUsers)
Ready
2
NURSE
By Medicine Cabinet
NURSE
(nurse,trays)
Enter EPR via Login Button
[loginAllowed nurse (compid,display, taskbar,users)]
(compid,display, removeLoginButton nurse taskbar, addUser nurse users)
1
Medicine Cabinet Computer
I/O
COMPUTER
(nurse,trays)
(compid,display,taskbar,users)
Leave Medicine Cabinet
(nurse,trays)
if loggedin nurse (compid,display,taskbar,users) then
(compid, blank, removeMedicineCabinetButtons nurse taskbar, removeUser nurse users)
else
(compid, display, removeMedicineCabinetButtons nurse taskbar, users)

Fig. 14.15 PourCheckTrays module

the medicine trays that this nurse currently has in possession. Initially, the nurses Jane Brown and Mary Green are ready (represented as tokens on the place Ready) and have no trays.

The occurrence of the transition ApproachMedicineCabinet models the situation where a nurse changes from being ready to being busy near the medicine cabinet. At the same time, two buttons are added to the taskbar of the medicine cabinet computer, namely one login button for the nurse and one patient list button for the nurse. In the CPN model, these taskbar buttons are added by the function `addMedicine-CabinetButtons`, which appears on the arc from the transition Approach-MedicineCabinet to the place MedicineCabinetComputer.

The possible actions for a nurse who is by the medicine cabinet are modelled by the three transitions PourCheckTray, EnterEPRviaLoginButton, and LeaveMedicine-Cabinet. Often, a nurse at the medicine cabinet wants to pour and/or check some trays. How this pouring and checking is carried out is modelled by the submodule PourCheckTray associated with the substitution transition PourCheckTray.

The transition EnterEPRviaLoginButton models the situation where a nurse clicks on the login button and makes a general-purpose login to EPR. It is outside the scope of the model to describe what the nurse subsequently does – the domain of the model is specifically medicine administration, not general EPR use. This transition has a guard which checks if a nurse is allowed to log into EPR. When a nurse logs in, the login button for that nurse is removed from the taskbar of the computer, modelled by the function `removeLoginButton`. Moreover, the nurse is added to the set of current users by the function `addUser`.

The transition LeaveMedicineCabinet models the effect of a nurse leaving: it is checked whether the nurse is currently logged in, modelled by the function loggedIn appearing in the if–then–else expression on the arc from the transition LeaveMedicineCabinet to the place MedicineCabinetComputer. If the nurse is logged in, the medicine cabinet computer automatically returns to a blank screen, removes the nurse's taskbar buttons (removeMedicineCabinetButtons), and logs him/her off (removeUser). If the nurse is not logged in, the buttons generated because of his/her presence are removed, but the state of the computer is otherwise left unaltered. In any case, the token corresponding to the nurse is put back on the place Ready.

14.2.3 Behavioural Visualisation of Medicine Administration

The interaction graphics built on top of the CPN model are shown in Fig. 14.16. The graphics are an interface to the CPN model, i.e., the interaction graphics are consistent with the CPN model and reflect the markings, transition occurrences, and marking changes that appear when the CPN model is executed, as explained in Chap. 13. The interaction graphics were added to the CPN model to support communication between the users (nurses) and the system developers, by reducing the distance between the CPN model and the users' conception of future work processes and their proposed computer support.

The graphics are divided into three windows. The Department window (at the top of Fig. 14.16) shows the layout of a hospital department, with wards, the medicine room, the 'team room' (the nurses' office), and two bathrooms. The Medicine room

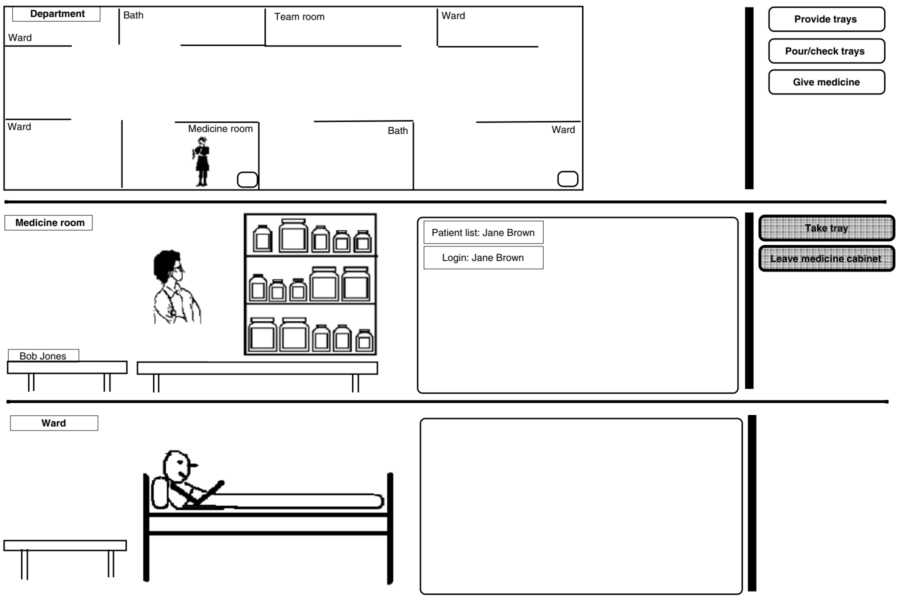


Fig. 14.16 Medicine administration interaction graphics

window (in the middle of Fig. 14.16) shows the medicine cabinet, pill boxes, tables, medicine trays, and a computer screen (enlarged). The Ward window (at the bottom of Fig. 14.16) shows a patient, a bed, a table, and a computer screen. Thus, the Department window gives an overview, and the other windows zoom in on areas of particular interest.

In Fig. 14.16, the graphics show a situation where nurse Jane Brown is in the medicine room, shown in the Department window and the Medicine room window, sufficiently close to produce two taskbar buttons on the computer. The user must make choices in order to drive the interaction graphics further. By selecting one of the grey buttons on the right in the Medicine room window, the user can choose to take a tray or leave the medicine room. The user can also select one of the taskbar buttons on the computer. These four choices correspond to enabled transitions in the CPN model. As an example, the user may push the LeaveMedicineCabinet button. This forces the transition with the same name in the CPN model (see Fig. 14.15) to occur. The result of the occurrence is experienced by the animation user, who sees Jane Brown walking away from the medicine cabinet and the removal from the computer screen of the taskbar buttons, which were generated because of Jane Brown's presence. If the animation user pushes the TakeTray button and then selects Bob Jones's medicine tray, this tray is moved close to the computer, and a medicine plan button for Bob Jones appears on the computer taskbar. If this button is pushed, the computer will display a screen similar to the one shown in Fig. 14.13.

14.2.4 Requirements Engineering for PHCS

When the PHCS project started, the first activities were domain analysis in the form of ethnographic field work, and a series of vision workshops with participation by nurses, doctors, computer scientists, and an anthropologist. One outcome of this analysis was natural-language descriptions of work processes and their proposed computer support. The first version of the CPN model presented in this section was based on these prose descriptions. The CPN model and the interaction graphics were extended and modified in a number of iterations, each version based on feedback about the previous versions. The interaction graphics served as a basis for discussions in evaluation workshops, with participation by nurses from hospitals in Aarhus and personnel from the software company involved.

Through the construction and use of the CPN model and the use of interaction graphics at the evaluation workshops, experience was gained in the use of CP-nets in requirements engineering. It could be observed that for PHCS, the CPN model and the interaction graphics were effective means for *specification*, *specification analysis*, *elicitation*, and *negotiation and agreement* of requirements, as discussed below.

The specification of requirements has a sound foundation because of the formality and unambiguity of the CPN model. In the case of the CPN model of medicine administration, there are requirements precisely described by the transitions that model manipulation of the computers involved. Each transition connected to a place that models a computer, for example, the place MedicineCabinetComputer shown in Fig. 14.15, must be taken into account. The following are examples of requirements induced by the transitions of the module in Fig. 14.15:

R1 When a nurse approaches the medicine cabinet, the medicine cabinet computer must add a login button and a patient list button for that nurse to the taskbar (transition ApproachMedicineCabinet).

R2 When a logged-in nurse leaves the medicine cabinet, the medicine cabinet computer must return to a blank display, remove the nurse's login button and patient list button from the taskbar, and log that nurse out (transition LeaveMedicineCabinet).

R3 When a nurse selects his/her login button, that nurse must be added as a user of EPR, and the login button must be removed from the taskbar of the computer (transition EnterEPRviaLoginButton).

Specification analysis is well supported by simulation, which allows experiments and trial-and-error investigations of various scenarios for the envisioned work process. Simulation combined with interaction graphics was considered the most appropriate means for specification analysis. It is easy for the nurses to understand, and the work processes can be modelled in as much detail as desired without worrying about state space explosion.

Elicitation includes the discovery of new requirements and the gaining of a better understanding of known requirements. Elicitation is, like specification analysis, well supported by simulation. Simulation spurs elicitation by triggering many questions. Simulation of a CPN model typically catalyses the participants' cognition and generates new ideas. Interaction with an executable model that is a coherent description of multiple scenarios is very likely to bring up new questions, and issues appear that the participants had not thought about earlier. Some examples of questions that appeared during simulation of the CPN model for medicine administration and their corresponding answers are:

Q1 What happens if two nurses are both close to the medicine cabinet computer?

A1 *The computer generates login buttons and patient list buttons for both of them.*

Q2 What happens when a nurse with several medicine trays approaches a bed?

A2 *In addition to a login button and a patient list button, only one medicine plan button is generated – a button for the patient associated with that bed.*

Q3 Is it possible for one nurse to acknowledge the pouring of medicine for a patient while another nurse at the same time acknowledges the giving of medicine to that same patient?

A3 *No, that would require more fine-grained concurrency control to be exercised over the patient records.*

Questions such as Q1–Q3 and their answers A1–A3 may imply changes to be made to the CPN model. As a concrete example, in an early version of the medicine administration CPN model, the leaving of any nurse from the medicine cabinet resulted in the computer display being blanked off. To be compliant with the principle of non-intrusive design for PHCS, the leaving of a nurse who is not logged in should of course not disturb another nurse who might be working at the computer, and the CPN model had to be changed accordingly.

Negotiation and agreement may be eased via CPN models. In large projects, negotiation about requirements inevitably takes place during the project. In many cases, this has strong economic consequences, because a requirements specification for a software system may be an essential part of a legal contract between, for example, a hospital and a software company. Therefore, it is important to be able to determine what requirements were included in the initial agreement. Questions such as Q1–Q3 above may easily be subject to dispute. However, if the parties involved have an agreement that medicine administration should be supported, and agree to the overall stipulation that the formal, unambiguous CPN model is the authoritative description, many disagreements can be quickly settled.

14.2.5 Conclusions from the PHCS Project

This project demonstrated that CPN models are able to support various requirements engineering activities. The CPN model and the interaction graphics can be seen as a supplement to UML use cases. Use cases describe work processes to be supported by a new IT system, and a set of use cases is interpreted as a set of functional requirements for that system. One of the main motivations for the requirements engineering approach chosen for PHCS was to build on top of prose descriptions of work processes and the proposed computer support, consolidated as UML use cases. The advantage of this was that the stakeholders of PHCS were already familiar with these UML use cases via the work on EPR. Having an executable representation of a work process supports specification analysis and elicitation, as we have discussed. The interaction graphics used in the project enabled users such as nurses and doctors to be actively engaged in specification analysis and elicitation, which is crucial. User participation increases the probability that a system is ultimately built that fits with the future users' work processes.

14.3 Embedded-System Design at Bang and Olufsen

This joint project [17, 75], conducted with Bang & Olufsen [3] was concerned with the design and analysis of the BeoLink system. A timed CPN model was developed, specifying the lock management subsystem which is responsible for the basic synchronisation of the devices in the BeoLink system. Methods based on state spaces, including a number of advanced state space methods, were used to verify the lock management system.

14.3.1 BeoLink System

The BeoLink system makes it possible to connect audio and video devices in a home via a dedicated network. A home equipped with the BeoLink system will typically have a number of audio/video sources such as radios, CD/DVD players, and TVs. Using the BeoLink system, it is possible to distribute these sources to different rooms. The CPN modelling and analysis focused on the *lock management protocol* of the BeoLink system. This protocol is used to grant devices exclusive access to services in the system, such as being able to use the loudspeakers when playing music. The lock management protocol is based on the notion of a *key*, and a device is required to possess a key to access services in the system. When the system is switched on, exactly one key must be generated by the devices currently in the system. Furthermore, this key must be generated within 2 seconds for the system to be properly working. Special devices in the system, called *audio* and *video masters*, are responsible for generating the key.

The MSC in Fig. 14.17 shows a typical communication sequence in a BeoLink system with four devices. A single User is present and wishes to change the CD track on Device1. The event key_wanted is sent to Device1, which is not currently the lock manager. Device1 therefore requests the key over the network by broadcasting a REQUEST_KEY *telegram* (message). Device3 is the lock manager and is ready to give away the key. Hence, Device3 sends a KEY_TRANSFER telegram to Device1 and the key is reserved. Device1 is granted the key upon reception of the KEY_TRANSFER telegram, and sends a telegram NEW_LOCK_MANAGER to Device3 as an acknowledgement of a successful key transfer. Finally, the User receives the event key_ready, and the change of track on the CD player can take place.

14.3.2 BeoLink CPN Model

Figure 14.18 shows the module hierarchy of the BeoLink CPN model. The submodule Network models the network that connects the devices in the system. The module Device and its submodules model the lock management protocol entities in each device. The submodules on the right, from RequestKey down to Function-

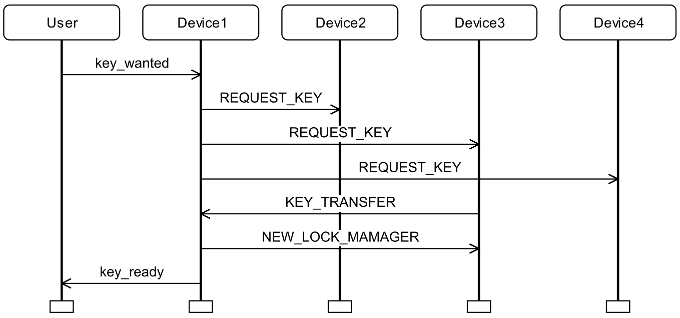


Fig. 14.17 Message sequence chart showing communication sequence in the BeoLink system

Lock2, correspond to the various functional blocks of the lock management protocol. The submodule KeyUser models the behaviour of devices as seen from the lock management protocol.

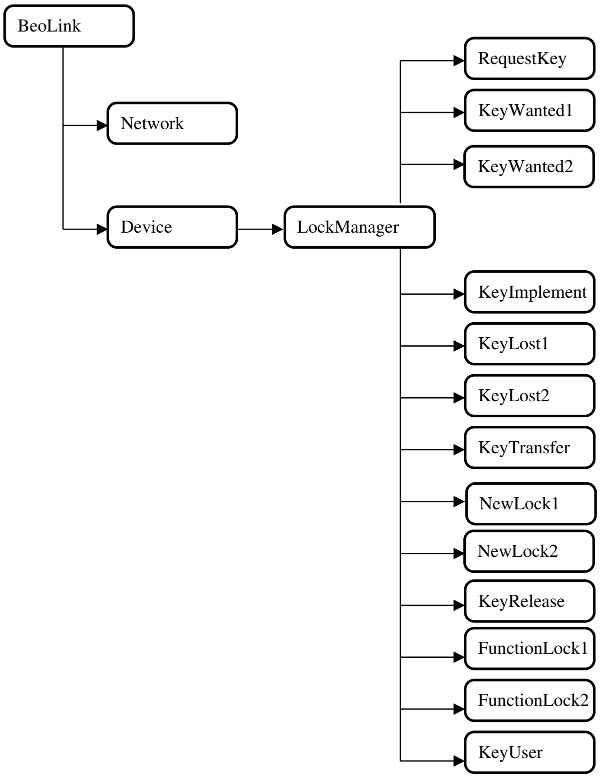


Fig. 14.18 Module hierarchy of the BeoLink model

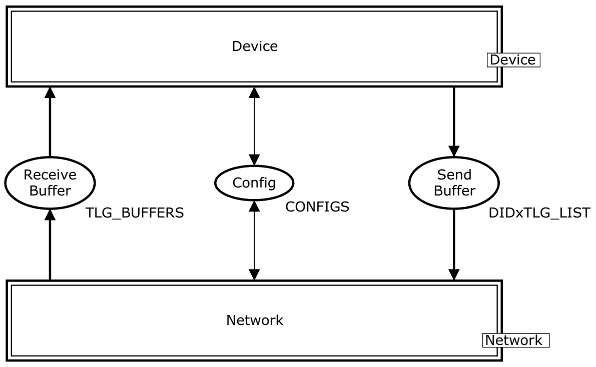


Fig. 14.19 BeoLink module

Figure 14.19 shows the BeoLink module. The substitution transition Network represents the network that connects the devices in the system. The substitution transition Device models the devices in the system. The CPN model provides a folded representation of the behaviour of the devices. This is achieved by encoding the identities of the devices as part of the colours of tokens (as in the protocol with multiple receivers in Sect. 5.4). This makes it possible to capture any number of devices without having to make changes to the net structure of the CPN model, and without having an instance of the submodules of the substitution transition Device for each of the devices in the system. This way of compactly representing any number of devices makes the CPN model parametric. The details will become evident when we present the KeyUser module.

The socket places ReceiveBuffer and SendBuffer in Fig. 14.19, which connect the two substitution transitions, model message buffers between the devices and the network. Messages in the lock management protocol are called telegrams and are abbreviated TLG. Each device has a buffer for outgoing and incoming telegrams. The place Config is used for the configuration of the CPN model.

The behaviour of devices, as seen from the lock management protocol, is modelled by the KeyUser module shown in Fig. 14.20. Each device has a cyclic control flow, where the device is initially idle (modelled by the place Idle), it then asks for the key (modelled by the transition RequestKey), and it enters a state where it is waiting for the key (modelled by the place Waiting). The granting of the key to a device is modelled by the transition GetKey which causes the device to enter a state where it is using the key (modelled by the place UseKey). When the device has finished using the key, it releases the key and returns to the idle state, where it may then ask for the key again. The places Status, Commands, and FunctionLockIn are used to model the internal state of a device. The places SendBuffer and ReceiveBuffer are linked to the accordingly named places in the BeoLink module via a sequence of port/socket relationships. The markings of these five places are also changed by the various functional blocks of the lock management protocol.

Figure 14.20 shows the initial marking of the CPN model, with three devices all in their idle state, as represented by the three tokens on the place Idle. A device is

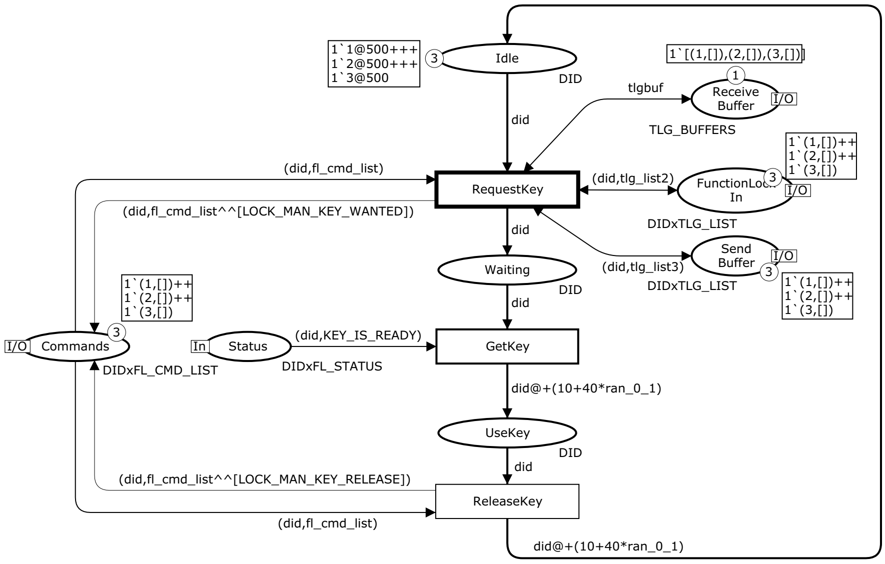


Fig. 14.20 Initial marking of the KeyUser module

identified simply by a number. In this marking any of the three devices may ask for the key, corresponding to the transition RequestKey being enabled with three different bindings depending on the device identifier assigned to the variable `did`. Figure 14.21 shows a marking of the KeyUser module where device 1 is using the key, whereas devices 2 and 3 have requested but not been granted the key.

The CPN model of the BeoLink system is timed. This means that the CPN model captures the time taken by the various events in the protocol. As an example, the transition GetKey uses the symbol `@+` in the arc expression on the output arc leading to the place UseKey. The number of time units to be added to the current model time is specified by the expression `10+40*ran_0_1`, where `ran_0_1` is a variable that can be bound to either 0 or 1. This models a situation where the event of obtaining the key can take either 10 or 50 time units.

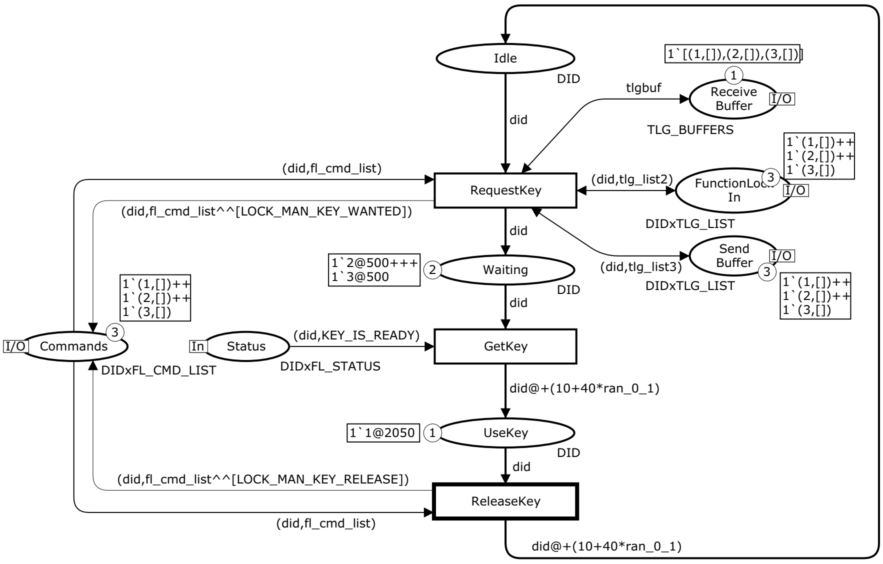


Fig. 14.21 Marking of the KeyUser module, where device 1 is using the key

14.3.3 State Space Analysis and Verification

The CPN model was first validated by means of simulation, and, later, state spaces were used to formally verify the properties of the BeoLink system. The three main correctness criteria of the lock management protocol are:

C1 *Key generation*. When the system is booted, a key must eventually be generated. The key is to be generated within 2.0 seconds.
C2 *Mutual exclusion*. At any time during the operation of the system at most one key exists.
C3 *Key access*. Any given device always has the possibility of obtaining the key, i.e., no device is ever excluded from getting access to the key.

Figure 14.22 shows an initial fragment of the state space for the BeoLink system. This contains the markings that are reachable from the initial marking by at most two transition occurrences. The initial marking (represented by node 1) was shown in Fig. 14.20. In this marking there are three enabled binding elements, since all three devices are able to request the key. The boxes positioned on top of the arcs describe the enabled binding element to which the arc corresponds by giving the transition name and the value bound to the variable `did` (the device identifier). The transition KeyWanted is in another module of the CPN model.

The state space of the timed BeoLink CPN model is infinite because the BeoLink system contains cyclic behaviour and because the absolute notion of time, as represented by the global clock and the timestamps of tokens, is carried over into the state

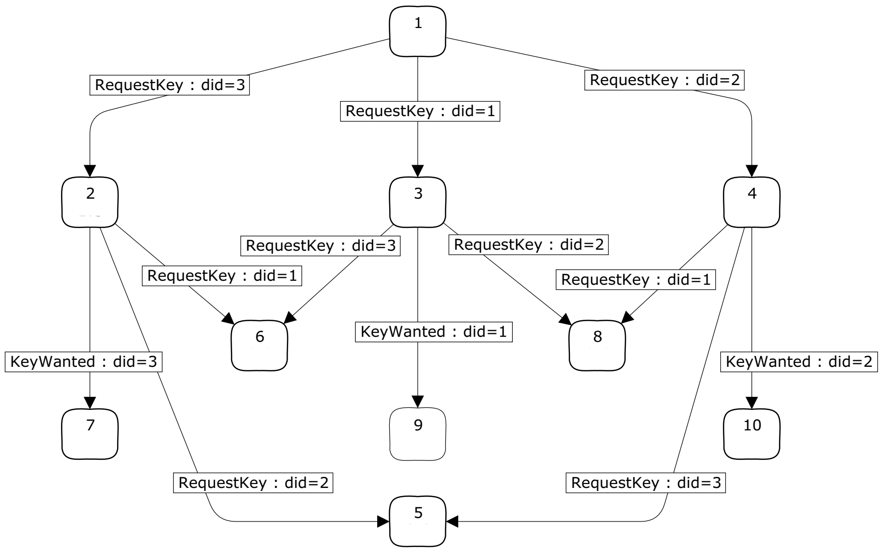


Fig. 14.22 Initial fragment of state space

space (see Sect. 10.3). Cyclic behaviour arises, for example, from the fact that devices may execute a loop where they request the key, are granted the key, and finally release the key. As a concrete example, consider the marking of the KeyUser module shown in Fig. 14.23. This marking is similar to the marking previously shown in Fig. 14.21, except that all devices have had the key once and device 1 now possesses the key again. The markings in Figs 14.21 and 14.23 are represented by two different nodes in the state space because the timestamps of the tokens and the values of the global clock differ.

The initial state space analysis of the CPN model considered the initialisation phase of the BeoLink system and the time-bounded *key generation* property C1. Verification of C1 was investigated by considering a partial state space, i.e., a finite fragment of the full state space. This partial state space was obtained by not generating successors for markings where the key had been generated or where the model time had passed 2 seconds. It was then checked that a key was present in the system in all markings for which successor markings had not been generated. To save computer memory, the arcs in the state space were not stored, since they were not needed for verifying the key generation property. Table 14.3 lists some statistics showing the number of nodes in the partial state space for different configurations of the BeoLink system. Configurations with one video master and a total of *n* devices are denoted VM:*n*, and configurations with one audio master and a total of *n* devices are denoted AM:*n*.

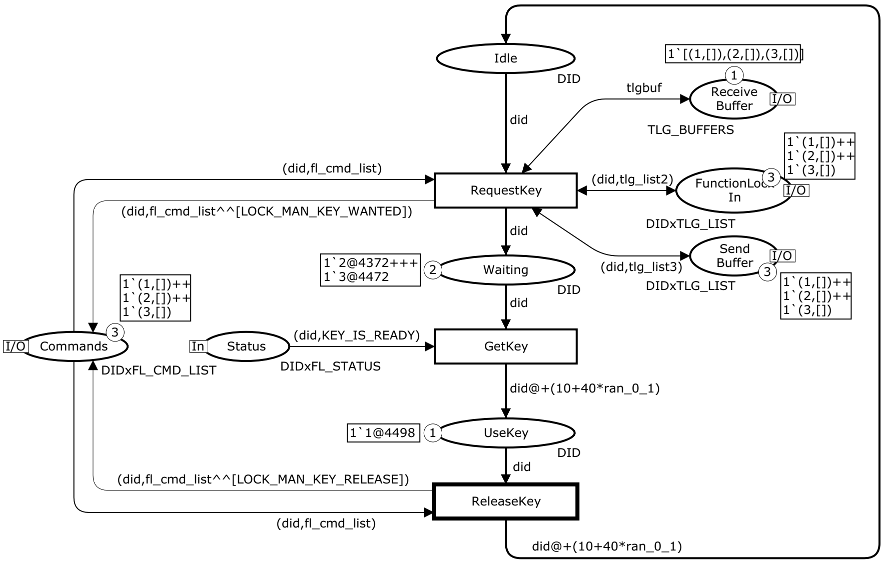


Fig. 14.23 KeyUser module, when all devices have used the key once

Table 14.3 Statistics for partial state space of the initialisation phase (global clock $\leq$ 2.0 seconds)

Configuration	Nodes
AM:3	1 839
AM:4	22 675
AM:5	282 399
VM:3	1 130
VM:4	13 421
VM:5	164 170

14.3.4 Application of Advanced State Space Methods

To conduct state space analysis of the full BeoLink system and not only the initialisation phase, the time equivalence method introduced in Sect. 10.4 was applied. This factors out the absolute notion of time and constructs a finite condensed state space whenever the state space of the underlying untimed CPN model is finite. Table 14.4 shows statistics for the condensed state space constructed using the time equivalence method. At the time of the project, it was not possible to generate the time-condensed state space for more than three devices with the available amount of computer memory. Using the condensed state space, it is now possible to verify also properties C2 (mutual exclusion) and C3 (key access). Property C2 can be expressed as the property that in no reachable marking is there more than one token on

Table 14.4 Statistics obtained with the time equivalence method for the full system

Config	Nodes	Arcs
AM:2	346	399
AM:3	27 246	37 625
VM:2	274	310
VM:3	10 713	14 917

the place UseKey (see Fig. 14.20), and property C3 can be expressed as the property that, from any reachable marking and for any device, it is always possible to reach a marking where the token corresponding to this device is on the place UseKey. These two properties can be checked using the standard query functions `PredAllNodes` and `HomePredicate` in the CPN state space tool.

The state space analysis presented above allowed only configurations with up to three devices to be verified because of the state explosion problem, i.e., the state spaces became too large to be computed with the available computer memory. To obtain state spaces for larger configurations, we applied the symmetry method (see Sect. 8.3) and the sweep-line method (see Sect. 8.2).

The symmetry method represents symmetric markings and symmetric binding elements using *equivalence classes*. The devices in the BeoLink system that are not audio or video masters are symmetric, in the sense that they behave in the same way with respect to the lock management protocol. They are distinguishable only by their device identity. This symmetry is also reflected in the state space (see Fig. 14.22). Consider, for instance, the two markings represented by nodes 2 and 4, which correspond to markings in which exactly one non-master device has requested the key (device 1 is the audio master in the configuration considered). These two markings are symmetric in the sense that node 2 can be obtained from node 4 by swapping the identities of devices 2 and 3. Similarly, the two states represented by node 7 and node 10 can be obtained from each other by interchanging the identity of devices 2 and 3. These two markings correspond to states in which one device has requested the key and the lock management protocol has registered the request. Furthermore, it can be observed that two symmetric states such as state 2 and state 4 have symmetric sets of enabled binding elements and symmetric sets of successor markings.

Figure 14.24 shows the initial fragment of the symmetry-condensed state space for the BeoLink system obtained by considering two markings equivalent if one can be obtained from the other by a permutation of the identities of the non-master devices. The nodes and arcs now represent equivalence classes of markings and binding elements, respectively. The equivalence class of markings represented by a node is listed in bracs in the inscription of the node; for example, node 2 represents markings 2 and 4 in Fig. 14.22. The basic idea of symmetry-condensed state spaces is to represent these equivalence classes by picking a representative for each equivalence class.

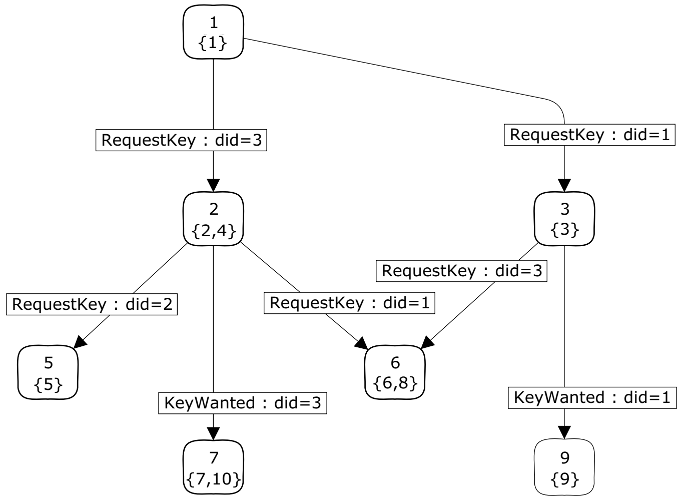


Fig. 14.24 Initial fragment of symmetry-condensed state space

Table 14.5 shows statistics obtained when using the symmetry method for the initialisation phase of the BeoLink system. The column 'State space nodes' lists the number of nodes in the full state space, and the column 'Symmetry nodes' lists the number of nodes in the symmetry-condensed state space. The column 'Node ratio' gives the number of nodes in the full state space divided by the number of nodes in the symmetry-condensed state space. The column 'Time ratio' gives the time used to generate the full state space divided by the time used to compute the symmetry condensed state space. The column '$(n-1)!$' lists the factorial of $n-1$, where n is the number of devices in the configuration. When there are n devices in the configuration, there are $(n-1)!$ possible permutations of the non-master devices. Hence, $(n-1)!$ is the theoretical upper limit on the reduction factor that can be obtained for a configuration with n devices. The computation time for symmetry-condensed state spaces becomes large for seven devices. This is due to the calculation of canonical

Table 14.5 Statistics for symmetry method: initialisation phase

Configuration	State space nodes	Symmetry nodes	Node ratio	Time ratio	$(n-1)!$
AM:3	1 839	968	1.9	1.0	2
AM:4	22 675	4 361	5.2	2.5	6
AM:5	282 399	15 865	17.8	10.0	24
AM:6	3 417 719	47 867	71.4	–	120
VM:3	1 130	594	1.9	1.0	2
VM:4	13 421	2 631	5.1	2.5	6
VM:5	164 170	9 328	17.6	10.0	24
VM:6	1 967 159	27 551	71.4	–	120
VM:7	22 892 208	68 683	333.3	–	720

representatives being costly (as described at the end of Sect. 8.3). The size of the full state space for the configurations AM:6, VM:6, and VM:7 has been calculated from the symmetry-condensed state space by computing the size of each equivalence class.

Table 14.6 lists statistics for the symmetry-condensed state spaces of the full BeoLink system. The column ‘Time equiv nodes’ gives the number of nodes in the state space obtained with the time equivalence method alone. The column ‘Symmetry + time equiv nodes’ gives the nodes for simultaneous use of the symmetry method and the time equivalence method. The number of nodes for the configurations AM:4 and VM:4 in the time equivalence method have been computed from the symmetry-condensed state spaces.

Table 14.6 Statistics for symmetry method: full system

Configuration	Time equiv nodes	Symmetry + time equiv nodes	Node ratio	Time ratio	$(n-1)!$
AM:3	27 246	13 650	1.92	2.0	2
AM:4	12 422 637	2 074 580	5.88	-	6
VM:3	10 713	5 420	1.98	2.0	2
VM:4	3 557 441	594 092	5.99	-	6

Next, we used the sweep-line method. The basic idea of the sweep-line method is to exploit a progress measure to explore all reachable markings of a CPN model, while storing only small fragments of the state space in memory at a time. This means that the peak memory usage is reduced. The sweep-line method is aimed at on-the-fly verification of safety properties, for example, determining whether a reachable marking satisfying a given state predicate exists. Hence, it can be used to verify properties C1 (key generation) and C2 (mutual exclusion) of the BeoLink system, but not property C3 (key access).

The global clock in a timed CPN model has the property that for two markings M and M', where M' is a successor marking of M, the value of the global clock in M is less than or equal to the value of the global clock in M'. This implies that the global clock can be used as a monotonic progress measure. Figure 14.25 shows how the markings/nodes in the state space fragment shown in Fig. 14.22 can be ordered according to this notion of progress. Markings in one layer all have the same value of the global clock. Layer 0 contains markings in which the global clock has the value 0. Layer 1 contains markings where the global clock is 500 time units. A marking in a given layer has successor markings either in the same layer or in a layer that represents further progress, but never in a layer that represents less progress.

Table 14.7 lists statistics for the application of the sweep-line method to the initialisation phase of the BeoLink system with the global clock as the progress measure.

To apply the sweep-line method to the full BeoLink system, we need to combine it with the time equivalence method (otherwise the state space will be infinite).

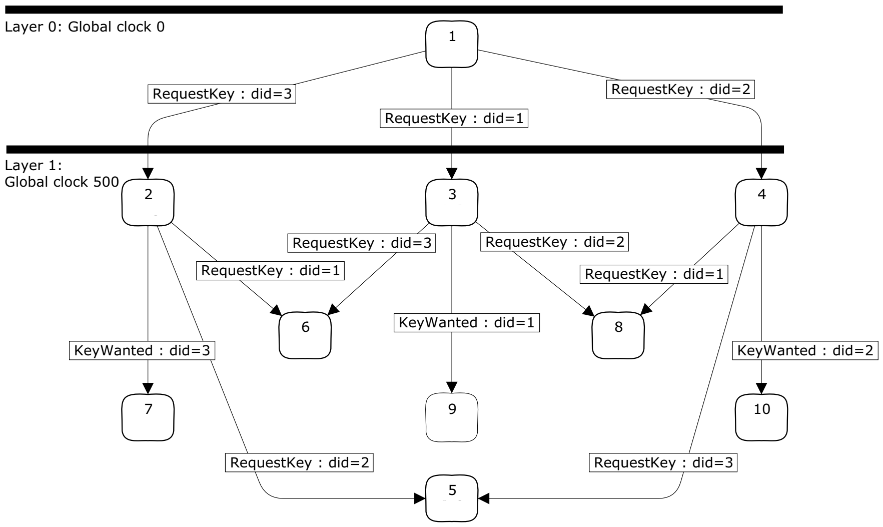


Fig. 14.25 Initial fragment of state space, arranged by progress

Table 14.7 Statistics for sweep-line method: initialisation phase

Configuration	State space nodes	Sweep-line peak nodes	Node ratio	Time ratio
AM:3	1 839	1 839	1.0	1.0
AM:4	22 675	5 169	4.4	1.2
AM:5	282 399	35 017	8.1	2.5
VM:3	1 130	1 130	1.0	1.0
VM:4	13 421	5 167	2.6	0.9
VM:5	164 170	34 968	4.7	2.2

The use of the time equivalence method implies that the global clock becomes 0 in all markings. It is, however, possible to define a non-monotonic progress measure based on the control flow of the devices and use this with the generalised sweep-line method [70]. The devices have a cyclic control flow where first they are idle, then they request the key, and finally they obtain the key. When they have used the key, they return to the idle state. This is a kind of local progress, starting from the idle state progressing towards the state where they have the key. This ordering can be used to define a non-monotonic progress measure. Details can be found in [70]. With this progress measure, the markings shown in Figs 14.21 and 14.23 have a higher progress value than the marking shown in Fig. 14.20. When a device releases the key and moves to the idle state, we have a regress arc in the state space (i.e., an arc along which the progress measure decreases).

Table 14.8 lists statistics for the application of the generalised sweep-line method to the full BeoLink system using the progress measure sketched above. The column 'Time equiv nodes' gives the number of nodes in the state space obtained with the time equivalence method alone. The column 'Nodes explored' lists the total number of nodes explored when the sweep-line method is used in combination with the time equivalence method, and the column 'Peak nodes' gives the peak number of nodes stored. It can be seen that some states are explored multiple times, which causes a time penalty. The sweep-line method achieves a reduction in peak memory usage to about 10%. The large time penalty was due to an inefficient implementation of deletion of states in the sweep-line library [43]. A more efficient algorithm for deletion of states has been developed in [71].

Table 14.8 Statistics for generalised sweep-line method: full system

Configuration	Time equiv nodes	Sweep-line + time equiv. Nodes explored	Sweep-line + time equiv. Peak nodes	Node ratio	Time ratio
AM:2	346	355	65	5.3	0.5
AM:3	27 246	28 363	2 643	10.3	0.3
VM:2	274	283	41	6.7	0.5
VM:3	10 713	11 388	1 039	10.3	0.5

We have seen above that it is possible to combine time-condensed state spaces with both the symmetry method and with the sweep-line method. It is also possible to use the sweep-line method and the symmetry method simultaneously. This combination was investigated in [8], where it was demonstrated that using the two methods simultaneously leads to a better reduction than when either method is used in isolation.

14.3.5 Conclusions from the BeoLink Project

This project demonstrated the use of CP-nets for modelling and validating a real-time system, i.e., a system where the correctness of the system depends on timing information. The construction of the CPN model was done in close cooperation between engineers at Bang & Olufsen and members of our research group. The engineers were given a four-day course on CP-nets, enabling them to construct large parts of the CPN model. This demonstrates (as also seen in other projects) that a relatively short introduction is required to get started on using CP-nets in industrial projects.

When the BeoLink project was originally conducted, only the initialisation phase of the lock management protocol was verified using state spaces [17]. The reason for this was that no advanced state space methods were available in the CPN computer tools at that time. Since then, a number of advanced state space methods have been developed and implemented, and the revised state space analysis in [75] has used these to verify configurations of the BeoLink system that could not be verified using ordinary state spaces. The application of the advanced state space methods demonstrated that these methods enable verification of larger configurations of a system, and in some cases allow the verification of all configurations that are expected to appear in practice. It was also demonstrated that two advanced state space methods can be used simultaneously to get a better reduction than obtainable from either method in isolation.

14.4 Scheduling Tool for Australian Defence Forces

This project [73, 74, 112], conducted with the Australian Defence Science and Technology Organisation (DSTO), was concerned with the development of a Course of Action Scheduling Tool (COAST). In the project, CPN modelling was used to conceptualise and formalise the planning domain to be supported by the tool. Furthermore, the CPN model constructed was extracted in executable form from CPN Tools and embedded into the server of COAST together with a number of tailored state space analysis algorithms. The project demonstrated how a CPN model can be used for the implementation of a computer tool by effectively bridging the gap between a design specified as a CPN model and the implementation of the system.

14.4.1 Plans and Task Schedules

A *plan* (also called a course of action) consists of a set of tasks. The key capability of COAST is the computation of *task schedules* (also called lines of operations). The tool supports the development and analysis of military plans and their task schedules. A CPN model is used to model the execution of tasks according to their preconditions and postconditions, the synchronisations imposed, and the resources available. The possible task schedules are then obtained by generating a state space for the CPN model and extracting paths from the state space leading from the initial marking to certain markings representing *end states*. The framework underlying COAST is based on four key concepts:

- *Tasks* are the basic units of a plan and have associated preconditions describing the conditions required for a task to start, and effects describing the results of its execution. A task also includes a specification of the resources required to execute the task, and may have a specified duration. Tasks also have other attributes, but these will be omitted in this presentation.
- *Conditions* are used to describe the explicit logical dependencies between tasks via preconditions and effects. As an example, a task `T1` may have an effect used

as a precondition of a task T2. Hence, T2 depends logically on T1 in the sense that it cannot be started until T1 has been executed.

- *Resources* are used by tasks during their execution. Resources typically represent aircrafts, ships, and personnel required to execute a task. Resources may be available only at certain times, for example owing to service intervals. Resources may be lost in the course of executing a task.
- *Synchronisations* can be used to capture requirements that a set of tasks must begin or end simultaneously, that there has to be a specific amount of time between the start and end of a certain task, and that a task can start only after a certain point in time. A set of tasks that are required to begin at the same time is said to be *begin-synchronised*. A set of tasks required to end at the same time is said to be *end-synchronised*. End-synchronisations can cause the duration of tasks to be extended.

Table 14.9 shows an example plan with six tasks. This table specifies for each task its preconditions, its effects, the required resources, and the duration of the task. In addition to the information provided in the table, the set $\{$T5, T6$\}$ of tasks are begin-synchronised and the set $\{$T4, T5, T6$\}$ of tasks are end-synchronised. The available resources are 4‘R1 ++ 3‘R2 ++ 3‘R3 ++ 1‘R4 ++ 1‘R5 (written as a multiset). Figure 14.26 provides a graphical illustration of the dependencies and synchronisations between the tasks, using dashed lines to indicate begin-synchronisations and end-synchronisations.

We want to calculate the possible task schedules, i.e., the ways in which the set of tasks can be sequenced. Each task schedule must respect the effects and preconditions, the available resources, and the synchronisation constraints. Figure 14.27 illustrates one such possible task schedule.

The COAST tool is based on a client–server architecture. The client constitutes the domain-specific graphical user interface and is used for the specification of plans. It supports the human planners in specifying tasks, resources, conditions, and synchronisations. To analyse a plan, this information is sent to the COAST server. The client can now invoke the analysis algorithms in the server to compute task schedules. The server also supports the client in exploring and debugging the plan in cases where an analysis shows that no task schedule exists. The communication

Table 14.9 Example plan with six tasks

Task	Preconditions	Effects	Resources	Duration
T1	–	E1	4‘R1	2
T2	E1	E2	2‘R2 ++ 2‘R3	4
T3	E1	E3	2‘R2 ++ 2‘R3	7
T4	E1	E4	1‘R2 ++ 1‘R3	–
T5	E2	E5	1‘R4	7
T6	E3	E6	1‘R5	7

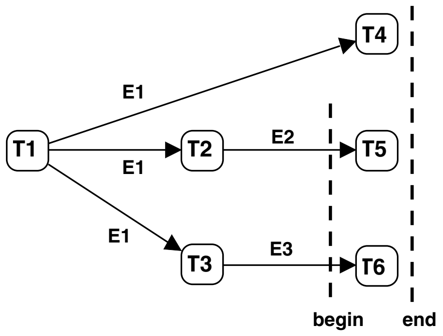


Fig. 14.26 Illustration of dependencies and synchronisations between tasks in the example plan

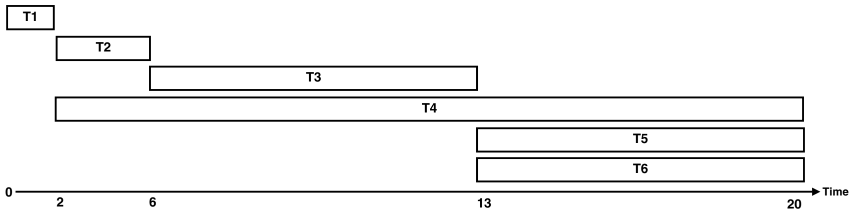


Fig. 14.27 One possible task schedule for the example plan

between the client and the server is based on a remote-procedure-call (RPC) mechanism implemented using the Comms/CPN library [42].

Figure 14.28 depicts the construction of the COAST server. The first step was to develop and formalise the planning domain, which provides the semantic foundation of COAST. This was done by constructing a CPN model that formally captures the semantics of tasks, conditions, resources, and synchronisations. This activity involved discussions with the prospective users of COAST (i.e., the military planners) to identify requirements and determine the concepts and working processes that were to be supported. The second step was to extract the constructed CPN model from CPN Tools. This was done by saving a *simulation image* containing the Standard ML code that CPN Tools generated for simulation of the CPN model. The CPN model is parameterised with respect to the set of tasks, conditions, resources, and synchronisations. This ensures that any given plan can be analysed by changing the initial marking (without changes to the net structure, arc inscriptions, or guards). This implies that the simulation image extracted from CPN Tools is able to simulate any plan, and hence CPN Tools was no longer needed once the simulation image had been extracted. The third step was the implementation of a suitable interface to the extracted CPN model and the implementation of the state space exploration algorithms.

The Model Interface component contains primitives that make it possible to set the initial marking of the CPN model to represent the concrete set of tasks, conditions, resources, and synchronisations constituting the plan to be analysed. In addi-

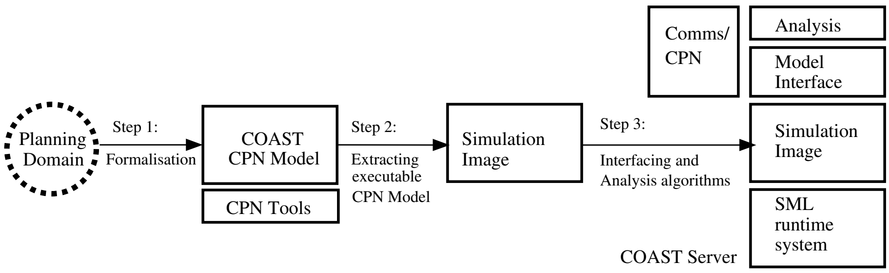


Fig. 14.28 Construction of the COAST server

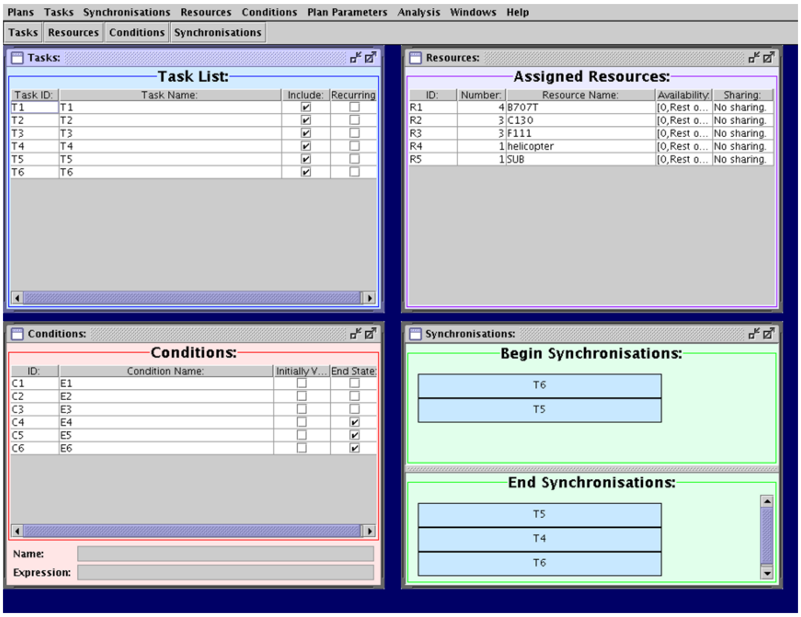
tion, it provides primitives that make it possible to obtain the set of enabled binding elements in a given marking, and the marking reached when an enabled binding element occurs. These primitives are used to implement the state space analysis algorithms in the Analysis component for task schedules. The Comms/CPN component was added, and it implements a remote-procedure-call mechanism that allows the client to invoke the primitives in the Analysis and the Model Interface components. The resulting application constitutes the COAST server.

Figure 14.29 shows a snapshot from the COAST client illustrating how the user views a plan in the editor. There are four main windows, showing the set of tasks, the assigned resources, the conditions, and the synchronisations. Figure 14.30 shows an example of how task schedules are reported to the user. It shows a task schedule which is identical to the schedule in Fig. 14.27, except that T3 now occurs before T2. The fact that the COAST server uses a CPN model as a basis for the scheduling analysis is fully transparent to an analyst using the COAST client.

14.4.2 COAST CPN Model

Figure 14.31 shows the module hierarchy for the CPN model. The CoastServer module is the top-level module in the CPN model, which consists of three main parts. The Execute module (left) and its submodules model the execution of tasks, i.e, the start, termination, abortion, and failure of tasks according to the set of tasks, resources, conditions, and synchronisations in the plan. The Environment module (middle) and its submodules model the environment in which tasks execute, and is responsible for managing the availability of resources over time, changes of conditions over time, and task failures. The Initialisation module (right) and its submodules are used for the initialisation of the model according to the concrete set of tasks, synchronisations, and resources in a plan. The CPN model is timed, since capturing the time taken by the execution of a task is an important part of the computation of task schedules.

Figure 14.32 lists the definitions of the colour sets that represent the key entities of a plan. A condition is modelled as a pair consisting of a `STRING`, specifying


Plans Tasks Synchronisations Resources Conditions Plan Parameters Analysis Windows Help
Tasks Resources Conditions Synchronisations
Tasks:
Resources:
Conditions:
Synchronisations:

Task List:

Task ID:	Task Name:	Include:	Recurring
T1	T1	✓	☐
T2	T2	✓	☐
T3	T3	✓	☐
T4	T4	✓	☐
T5	T5	✓	☐
T6	T6	✓	☐

Assigned Resources:

ID:	Number:	Resource Name:	Availability:	Sharing:
R1	4	B707T	[0,Rest o...	No sharing.
R2	3	C130	[0,Rest o...	No sharing.
R3	3	F111	[0,Rest o...	No sharing.
R4	1	helicopter	[0,Rest o...	No sharing.
R5	1	SUB	[0,Rest o...	No sharing.

Conditions:

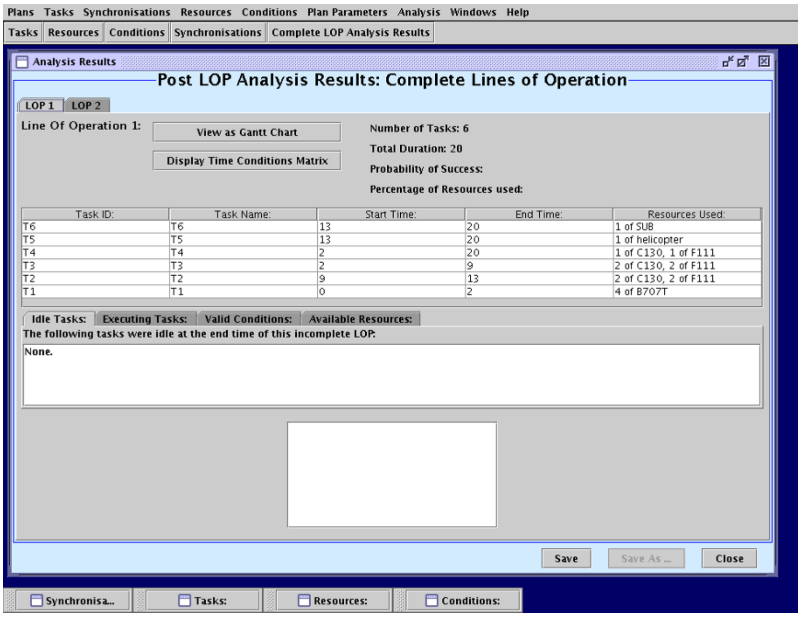
ID:	Condition Name:	Initially V...	End State:
C1	E1	☐	☐
C2	E2	☐	☐
C3	E3	☐	☐
C4	E4	☐	✓
C5	E5	☐	✓
C6	E6	☐	✓

Name:
Expression:

Begin Synchronisations:
T6
T5

End Synchronisations:
T5
T4
T6

Fig. 14.29 Snapshot from editing a plan in the COAST client


Plans Tasks Synchronisations Resources Conditions Plan Parameters Analysis Windows Help
Tasks Resources Conditions Synchronisations Complete LOP Analysis Results
Analysis Results
LOP 1 LOP 2
View as Gantt Chart
Display Time Conditions Matrix
Idle Tasks: Executing Tasks: Valid Conditions: Available Resources:
Save Save As ... Close
Synchronisa... Tasks: Resources: Conditions:

Post LOP Analysis Results: Complete Lines of Operation

Line Of Operation 1:

Number of Tasks: 6
Total Duration: 20
Probability of Success:
Percentage of Resources used:

Task ID:	Task Name:	Start Time:	End Time:	Resources Used:
T6	T6	13	20	1 of SUB
T5	T5	13	20	1 of helicopter
T4	T4	2	20	1 of C130, 1 of F111
T3	T3	2	9	2 of C130, 2 of F111
T2	T2	9	13	2 of C130, 2 of F111
T1	T1	0	2	4 of B707T

The following tasks were idle at the end time of this incomplete LOP:
None.

Fig. 14.30 Snapshot from analysing a plan in the COAST client

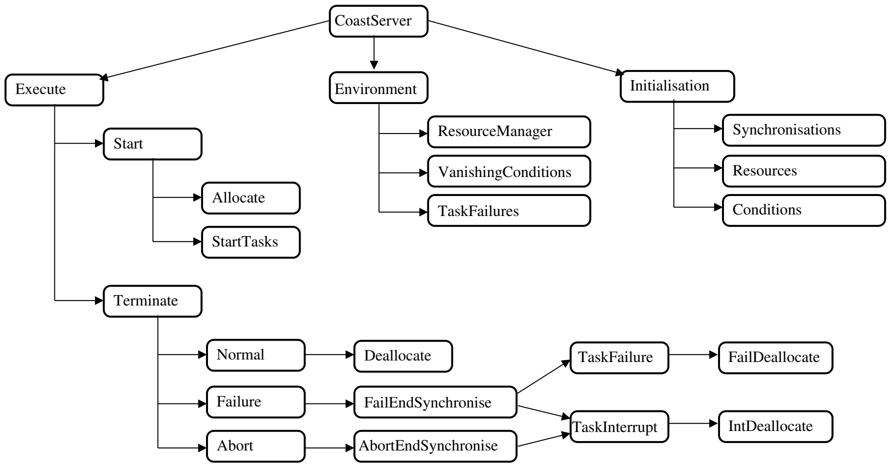


Fig. 14.31 Module hierarchy of the COAST model

the name of the condition, and a boolean, specifying the truth value. The colour set `ResourceSpecification` is used to represent the state of the resources assigned to the plan. The colour set `Resources` is defined as a `union` and is used for modelling the idle and lost resources. The assigned resources also have a specification of the *availability* of the resources (via the colour set `Availability`), specifying the time intervals during which or the start time at which the resource is available.

Tasks are the executable entities in a plan. They are modelled by the colour set `Task`, which is defined as a record consisting of 11 fields. The `name` field is used to specify the name of the task, and the `duration` field is used to specify the minimal duration of the task. The duration of a task may be extended owing to synchronisations, and not all tasks are required to have a specified minimal duration, since their durations may be given implicitly by synchronisations and conditions (see T4 in Table 14.9). The remaining fields can be divided into:

- *Preconditions*, which specify the conditions that must be valid before the task is started. The colour set `Conditions` is used for modelling the condition attributes of tasks. The `normalpreconditions` specify the conditions that must be satisfied for the task to start. A subset of the normal preconditions may be further specified as `vanishingpreconditions` to represent the effect that the start of the task will invalidate such preconditions. The `sustainingpre-conditions` specify the set of conditions that must be satisfied for the entire duration of the execution of the task. If a sustaining precondition becomes invalid, then it will cause the task to *abort*, which may in turn cause other tasks to be *interrupted*. The `terminationpreconditions` specify the conditions that must be satisfied for the task to terminate.
- *Effects*, which specify the effects of starting and executing the task. The `instanteffects` are conditions that become immediately valid when the

```
colset Condition   = product STRING * BOOL;
colset Conditions  = list Condition;

colset Resource     = product INT * STRING;
colset ResourceList = list Resource;

colset AvailSpecification = union INT : INTxINT + FROM : INT;
colset Availability       = list AvailSpecification;

colset ResourcexAvailability = product Resource * Availability;
colset ResourceSpecification = list ResourcexAvailability;

colset Resources = union IDLE : ResourceSpecification
                       + LOST : ResourceSpecification;

colset Task = record
    name           : STRING *
    duration       : Duration    *
    normalpreconditions      : Conditions *
    vanishingpreconditions   : Conditions *
    sustainingpreconditions  : Conditions *
    terminationpreconditions : Conditions *
    instanteffects           : Conditions *
    posteffects              : Conditions *
    sustainingeffect         : Conditions *
    startresources : ResourceList *
    resourceloss   : ResourceList;

colset BeginSynchronisation = list Task;
colset EndSynchronisation   = list Task;
```

Fig. 14.32 Colour set definitions for planning

task starts executing. The `posteffects` are conditions that become valid at the moment the task terminates. Finally, `sustainingeffects` are conditions that are valid as long as the task is executing.

- *Resources*, which specify the resources required by the task during its execution. Each resource is modelled by the colour set `Resource`, which is a product of an integer (`INT`), specifying the quantity, and a string (`STRING`), specifying the resource name. Resources may be lost or consumed in the course of executing a task. The `startresources` are resources required to start the task, and they are allocated for as long as the task is executing. The `resourceloss` are resources that may be lost during execution of the task.

The colour sets `BeginSynchronisation` and `EndSynchronisation` are used to specify that certain tasks have to begin or end at the same time.

Figure 14.33 shows the top-level module of the CPN model. It contains three substitution transitions and four places. The place Resources models the state of the resources, and the place Conditions models the values of the conditions. The place Idle contains the tasks that are yet to be executed, and the place Executing contains the tasks currently being executed. The marking in Fig. 14.33 represents an intermediate state in the execution of the the plan shown in Table 14.9. The place Conditions contains one token, which is a list containing the conditions in the plan and their truth values. The colour set for the places Resources, Executing, and Idle are complex. Hence, we have shown only the numbers of tokens and not the colours. The latter two places contain a token for each task which is Idle and Executing, respectively. The place Resources contains two tokens. One of these is a list describing the current set of idle (available) resources. The other token is a list describing the resources that have been lost up to now.

Figure 14.34 shows the Allocate module, which is one of the submodules of the substitution transition Execute (see Fig. 14.33). This module represents one of the steps in starting tasks. The transition Start models the start of a set of begin-synchronised tasks. The two port places Resources and Conditions are associated with the accordingly named places of the top-level module shown in Fig. 14.33 via a sequence of port–socket relations. An occurrence of the transition removes a token representing the begin-synchronised tasks (assigned to the variable `tasks`) from the place Tasks, a token representing the idle resources (bound to the variable `idleres`) from the place Resources, and a token representing the values of the conditions (bound to the variable `conditions`) from the place Conditions. The transition adds a token representing the set of tasks to be started to the place Starting and puts a token back on the place Conditions, updated according to the instant effects of the tasks. All idle resources are put back on place Resources, since the actual allocation is done in a subsequent step modelled by another module. The guard checks that the preconditions of the tasks are satisfied and that the necessary resources are available.

Other modules model the details of task execution and their effects on conditions and resources. They have a complexity similar to the Allocate module.

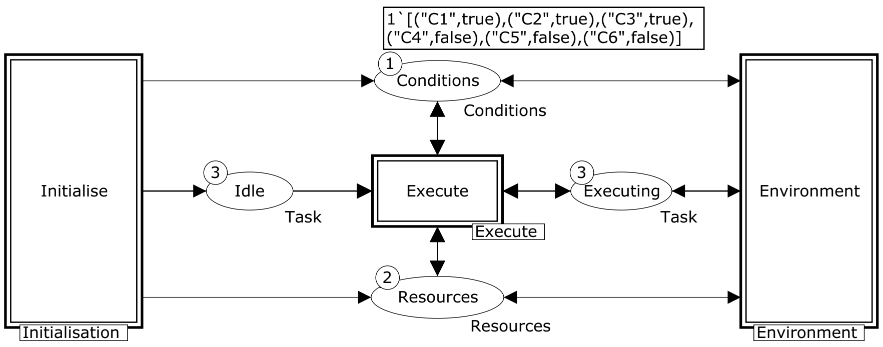


Fig. 14.33 CoastServer module

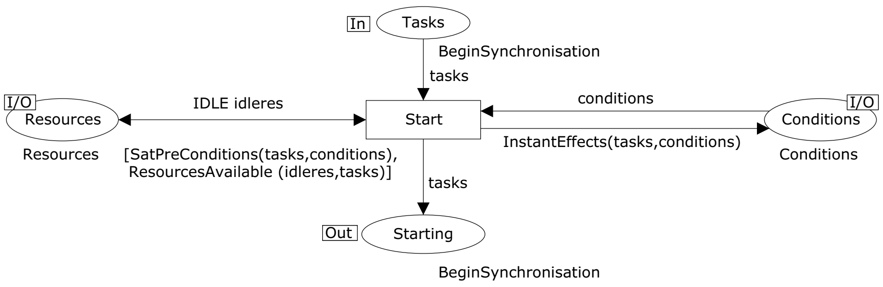


Fig. 14.34 Allocate module for starting tasks

14.4.3 Generation of Task Schedules

The main analysis capability of COAST is the generation of task schedules, i.e., a specification of start and end times for the tasks in a plan. The process of task schedule generation consists of two phases. In the first phase, a state space is generated relative to the plan to be analysed. Successors are not generated for states that qualify as desired end states according to the conditions specified by the user. In the second phase, the task schedules are computed by traversing the constructed state space. They are determined from the paths in the state space, and are divided into two classes. *Complete task schedules* are schedules that lead from the initial marking to a marking representing a desired end state. *Incomplete task schedules* are those that lead to markings representing undesired end states, i.e., dead markings that do not satisfy the conditions specified by the user. When incomplete schedules are reported, the user will typically investigate the causes of these using queries about tasks, conditions, and resources in various states. In that sense, COAST also supports the planner in identifying errors and inconsistencies in the plan under analysis.

Figure 14.35 shows the state space for the example plan shown in Table 14.9. Node 1, on the left, corresponds to the initial marking. The thick arcs in the state space correspond to the start and termination of tasks. The other arcs correspond to internal events in the CPN model. The thick arcs have labels of the form $Si : t \text{ or } Ei : t$, where i specifies the task number and t specifies the time at which the event takes place. As an example, task T1 starts at time 0, as specified by the label on the outgoing arc from node 1, and ends at time 2, as specified by the label on the outgoing arc from node 2.

The computation of task schedules is based on a breadth-first traversal of the state space starting from the initial marking. The basic idea is to compute the schedules leading to each of the markings encountered during the traversal of the state space, where the schedules for a given marking are computed from the schedules associated with its predecessor markings. The algorithm exploits the fact that the state space of the CPN model is acyclic for any plan, and that the paths leading to a given marking in the state space all have the same length.

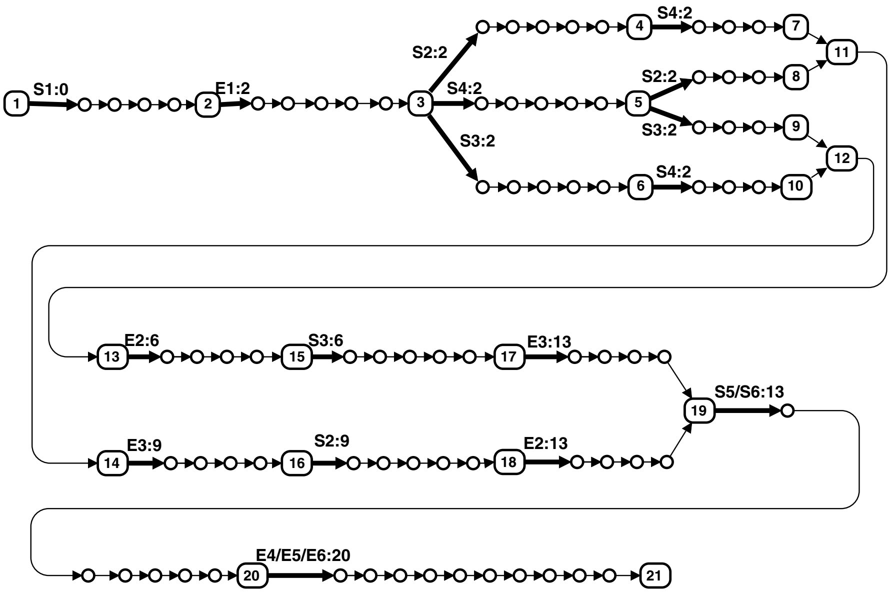


Fig. 14.35 State space for the example plan

We shall now illustrate how the algorithm operates. Figure 14.36 shows the task schedule information associated with each marking in the first part of the state space. The only schedule associated with the initial marking is the empty task schedule, represented by the empty list [ ]. Task schedules for the successor marking of the initial marking are now computed. The outgoing arc from node 1 corresponds to the start of a task. Hence, the schedule is augmented with information about the time at which T1 was started. This results in the schedule [(T1, 0, ?)]. The schedule remains the same until the arc corresponding to the termination of T1 at time 2 is reached. Then, the termination time of T1 is recorded in the schedule [T1, 0, 2]. The new schedule is propagated forwards and when node 3 is reached, the schedule is propagated along three branches corresponding to the three successor markings of node 3. The generation of schedules continues until nodes 7, 8, 9, and 10 are reached. Here the schedules associated with nodes 7 and 8 are merged and associated with node 11, since the start times and termination times of each of the tasks in the schedules are identical. Similarly, the schedules associated with nodes 9 and 10 are merged and associated with node 12. The breadth-first traversal now continues until, eventually, node 21 in Fig. 14.37 is reached, where the two complete schedules leading to the desired end state have been computed. The first schedule corresponds to the one shown in Fig. 14.27, and the second corresponds to the one shown in Fig. 14.30.

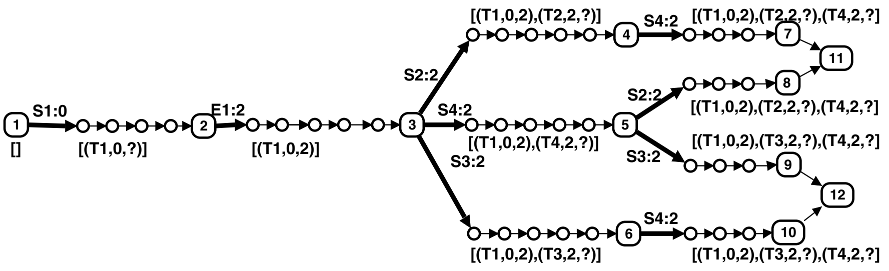
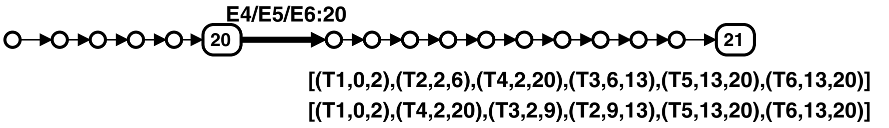


Fig. 14.36 Start of task schedule generation



E4/E5/E6:20
[(T1,0,2),(T2,2,6),(T4,2,20),(T3,6,13),(T5,13,20),(T6,13,20)]
[(T1,0,2),(T4,2,20),(T3,2,9),(T2,9,13),(T5,13,20),(T6,13,20)]

Fig. 14.37 Termination of task schedule generation

The typical planning problems to which COAST is applied consist of 15–25 tasks, resulting in state spaces with 10–20,000 nodes and 25–35,000 arcs. The state spaces are relatively small because the conditions, available resources, and imposed synchronisations strongly limit the possible orders in which the tasks can be executed.

14.4.4 Conclusions from the COAST Project

The role of CP-nets in the development of COAST was threefold. Firstly, CPN modelling was used in the development and specification of the underlying framework. Secondly, the CPN model constructed was used directly in the implementation of COAST by embedding it into the COAST server, which constitutes the computational back end of COAST. Hence, CP-nets provide a semantic foundation by formalising and implementing the abstract conceptual framework underlying the tool. Finally, the analysis capabilities of COAST are based on state space methods.

The development of the COAST tool is an example of how the usual gap between the design, as specified by a CPN model, and the final implementation of a system can be overcome. The CPN model that was constructed to develop the conceptual and semantic foundation of COAST is being used directly in the final implementation of the COAST server. The project demonstrates the value of having a full programming-language environment in the form of the Standard ML compiler integrated into CPN Tools. Standard ML was crucial in several ways for the development of COAST. It allowed a highly compact and parameterisable CPN model to be constructed, and also allowed the CPN model to become the implementation of the COAST server. The parameterisation is important for ensuring that the COAST server is able to analyse any set of tasks, conditions, resources, and synchronisations without the user having to make changes to the CPN model. Having a full programming language available also made it possible to extend the COAST server with the specialised algorithms required to extract the task schedules from the generated state spaces.