

Designing a Security System by means of Coloured Petri Nets

Jens Linneberg Rasmussen and Mejar Singh

Computer Science Department, Aarhus University
Ny Munkegade, Bldg 540, DK-8000 Aarhus C, Denmark
E-mail: {kaj eni ka, si ngh}@dai mi . aau . dk

Abstract. In this paper, we present an industrial use of Coloured Petri Nets (CP-nets) in designing a security system. An animation utility was developed which made it possible to perform user-friendly CP-net simulations. Furthermore, occurrence graphs (also known as reachability graphs and state spaces) were used for debugging the CP-net. In this way, a series of errors in the model were found and corrected. The CP-net design is used as a specification of the implementation of the security system. Therefore, finding errors by means of simulations and occurrence graph analysis reduces the amount of errors in the final implementation – making the software quality higher, which is the goal of the project.

1 Introduction

In the process of developing a new version of a security system at Dalcotech A/S¹, CP-nets were used for designing the software of the system. From the initial sketches of the system to the final system specification, CP-nets were applied. An animation utility was developed which made it possible to perform user-friendly CP-net simulations. Occurrence graphs were generated from the CP-net. They were used for investigating the dynamic properties of the CP-net and a series of errors in the model were located. The project demonstrates how the simulation and analysis methods associated with CP-nets are valuable when designing industrial systems.

We begin with introducing the security system and the project organisation. After this, we focus on the process of modelling the system and describe the final model. Additionally, we describe the use of the animation utility in simulations and the experiences gathered through occurrence graph analysis. Finally, we look at the pros and cons of using CP-nets for designing an industrial system.

Security System

This presentation of the PRISMA C-96 security system which is being developed by Dalcotech is based on [Dal95]. The security system will be the successor

¹ Dalcotech A/S is an engineering company which produces electronic systems, e.g. security systems. The company resides at: Bouet Møllevæg 16, 9400 Nørresundby, Denmark, tlf: +45 98191799, fax: +45 98190799.

of PRISMA C-91 (described in [Dal94]), but the software is constructed from scratch and the functionality of the system is extended. The security system is an intruder alarm system with advanced functionality and with the capability of handling large installations. In Fig. 1, a small example of an installation is given [Dal95].

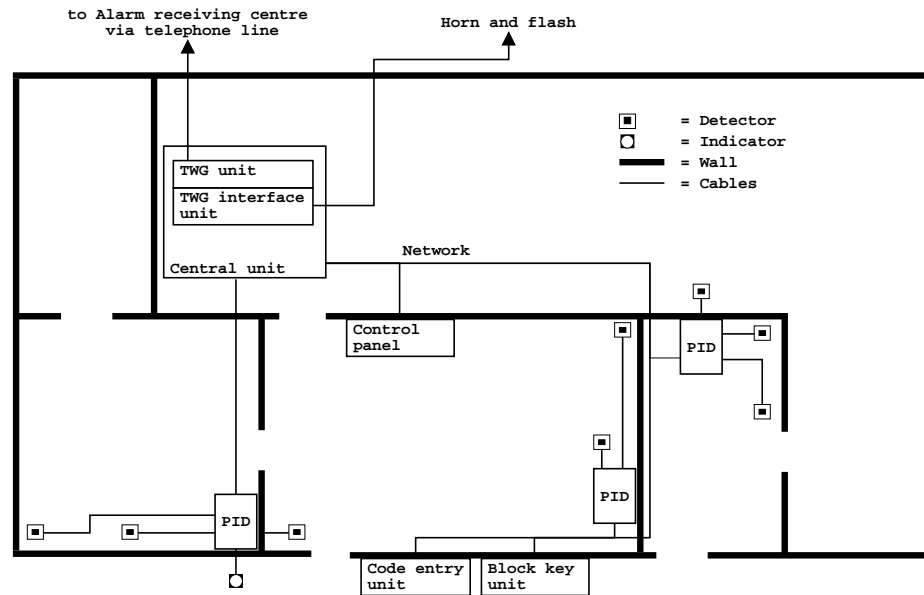


Fig. 1. PRISMA C-96 in a small installation

The *central unit* controls the security system and is connected to an alarm receiving centre via the public telephone system. Normally, an installation needs only one central unit. The PIDs (PRISMA Interface DeVices) handle physical *input* (e.g. detectors) and *output* (e.g. light-indicators and horns). There can be several PIDs in an installation, and these are connected to the central unit by a network. The system is operated primarily from the *control panels* located inside the building(s). The control panels are equipped with a keypad and an alphanumeric display. They can be used for retrieving information on the condition of the system and for controlling it, e.g. by acknowledging alarms. Access to a building (and the control panels inside) is granted by presenting a valid code, key, or magnetic card (or combinations of these) at an *entry unit*. Generally, entry units are located on the exterior wall of the supervised building(s), close to the entrance. These units can be code entry units, block key units, or magnetic card readers.

The security system divides the building(s) into areas, which may overlap each other. Generally speaking, there are two states for an area: It can be *set*, meaning that the activation of a detector in the area will lead to an alarm. The alarm will be reported locally, by means of horns and light-indicators, and

externally, by sending a message to the alarm receiving centre. Alternatively, an area can be *unset*. In this state, activation of a detector in the area will not cause an alarm. However, special input (e.g. glass-break detectors) can cause an alarm even if the corresponding area is unset. The states of the areas are controlled by the users via control panels and entry units. The most important function of the security system is to report an alarm whenever a detector is activated in a set area or under other illegal circumstances.

Installations can be quite large – the system handles up to a maximum of 100 areas, 1000 input, 1000 output, 1000 users (with different access codes), etc. The reliability of the system is essential, which is one of the reasons for applying CP-nets. The aim of the project is to increase the quality of software and make the new system more flexible than the old system.

Project Organisation

The project of using CP-nets for designing the security system was conducted by 8 people. Along with 3 security system engineers from Dalcotech we² participated in the *work group* which performed the actual modelling of the system. Additionally, 3 *consultants* participated in monthly project meetings with the work group. The consultants revised the work group's CP-nets, constructed design proposals, and were responsible for introducing the security system engineers to CP-nets. The introduction of CP-nets in the company is financially supported by ESSI – an EU-funded program meant to improve the quality of software production in Europe. The expenses for consultants, new machinery, CP-net tool, etc. were financed by this program.

2 Initial Models

During the first two months of the project, the three employees at Dalcotech became acquainted with CP-nets [Jen92] and with Design/CPN [CPN93] which was used as CP-net tool. Firstly, they attended an introductory course about developing CP-nets. After the course, the security system engineers continued developing CP-nets in connection with some small projects. Meanwhile, the system design was considered.

At this stage, it was discussed how CP-nets were to be applied in the project. What was the purpose of the model? Which level of detail should the model describe? Which parts of the system should be modelled? The work group found that the task was to design the software of the essential part of the security system: the central unit. Communication protocols, hardware failures, or the individual low-level behaviour of each control panel and entry unit were not to be modelled. On the other hand, the model should be sufficiently detailed. For instance, it should describe that some of the actions in the system are based on

² In this paper, *we* refers to us (the authors) alone. The work was done in connection with our master's thesis.

time-events. As an example, a user has to insert his key in the block key unit within 60 seconds after entering his code at the code entry unit. At this stage, the work group did not know how to model this; it was just recognised that time-events were an important factor of the system.

During the startup period the security system engineers realised that CP-nets were well suited for making detailed system designs. This was reflected by the use of CP-nets in connection with two other projects at Dalcotech. With respect to the security system model, the startup phase did not yield many tangible results but provided an increased understanding of how CP-nets were to be used. The startup phase lasted 2-3 months, partly because it takes a while to become familiar with the use of CP-nets, and partly because in this phase, the work group reflected upon the system's functionality. The work group had to set some goals for what to achieve with the CP-net. A *requirements specification* was initiated by the security system engineers. This was a textual description of the future system and was to serve as a guideline for developing the CP-net.

The development of the CP-net was definitely going to be part of a process that would lead to an implementation of the system. One of the consultants suggested investigating whether it could be possible to automatically generate code from the CP-net. The feasibility of this new idea was examined in connection with the project. However, at this stage it was uncertain whether it was possible to perform automatic code generation, or whether the CP-net was to serve as a template for a C/C++ implementation of the system.

Client/Server Model

With a possible C/C++ implementation in mind, the work group started modelling the system with a *client/server* approach, as suggested by Dalcotech. The work group found that it would ease implementation if the model described a client/server system. Also, the client/server approach is very flexible, allowing expansion of the system with few changes to old parts.

In this CP-net, a single place (containing message queues) models the process communication between clients and servers, each of which is modelled by a subnet. Data in one of the subnets of the client/server model can only be modified in the same subnet, which inhibits situations in which two processes try to modify the same data at the same time. In this way, a high degree of data control and modularity is obtained. On the other hand, every subnet (process) will need to communicate through the place containing the message queues in order to access data which is situated in other subnets. This induces a complex message structure used for communication.

During the startup phase, one of the consultants constructed an alternative CP-net, describing the security system. This model was also developed further while the client/server model was being constructed. The purpose of this model was to try to capture the essence of the system as described in the requirements specification of that time. The implementation of the model was not considered and therefore, we refer to this model as the *implementation independent*

model. This model had a clearer structure which did not force all communication through one place. The main drawback of this model was that the final system would be less flexible (harder to expand). It seemed that the client/server approach made it easier to add an extra process or change a specific process later on. Furthermore, the security system engineers felt that the client/server model would be easier to implement because it consisted of well defined modules and message structures.

In April '95 it was decided that *one* approach had to be chosen and used for the rest of the project. The choice fell on the implementation independent approach. The loss of clarity and the focus on implementation inherent in the client/server approach were too serious drawbacks. If automatic code generation was to be applied, the implementation benefits from a client/server model would be insignificant. The bottom line is that *it was a question of modelling the security system concept instead of modelling an implementation of a security system*. It was found that a more comprehensible and general model would be desirable in the design of the system. Also, the employees at Dalcotech were becoming more acquainted with CP-nets and beginning to get an idea of how an implementation independent model could be implemented – even if automatic code generation was not to be applied. As a result of this decision, a new model was constructed. Parts of the previous models could be reused, but as a matter of fact this new model was developed almost from scratch, with major revisions of all data structures and functions previously declared.

3 Final Model

Having already made two (incomplete) models of the system, it was much easier to start on the new model. The work group was very careful with designing the fundamental structure of the CP-net. Quite some time was spent on identifying candidates of the system's processes and defining colour sets which were as comprehensible as possible, based on the requirements specification of that time. The work group used a *top-down* approach and focussed on determining the basic structure of the CP-net.

In Fig. 2, the top page of the CP-net is depicted. The main system components are the three substitution transitions³ – each representing a subnet of the model. The Environment models the peripheral devices; the Central unit forms the main part of the model, it manages the global data and models the functionality of the security system; and the Configuration initialises the global data, by loading data from configuration-files.

On the places in the left column, the communication between the Central unit and the external devices present in the Environment takes place. These places will be referred to as interface places, as they constitute the I/O interface to the Environment. For instance, the digital input/output is modelled by two places. The Changed Logical input place holds a list of the changed logical input (e.g. detectors) reported by PIDs in the Environment. The Changed output

³ Substitution transitions are marked with an HS-tag.

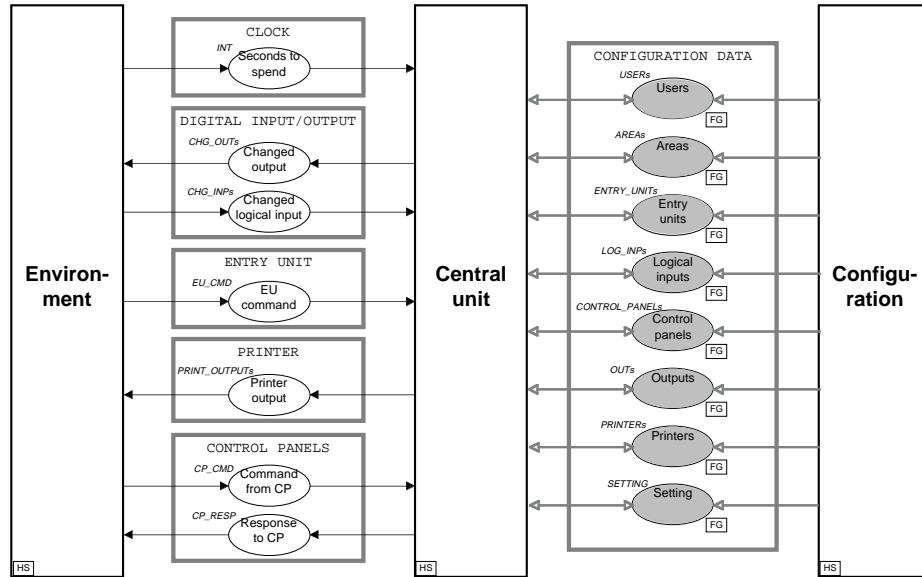


Fig. 2. Top level of the CP-net

place holds a list of changed output (e.g. light-indicators), which is sent by the Central unit to PIDs.

The shaded places in the right column represent global data initialised from the Configuration. Their colour sets are lists of records; each record has between 10 and 20 entries describing both static and dynamic data. The shaded places, marked by FG-tags, are all fusion places. A fusion place is a member of a set of places (called a fusion set), which conceptually represents one single place. The interface places are sockets. Sockets define the interface to substitution transitions; each socket is related to a port place on the subpage.

Splitting the model into Environment, Central unit, and Configuration subnets turned out to be a good design decision, as it isolates the software of the central unit from the peripheral hardware units and the configuration. It also allows three different versions of the environment and configuration: one set of subnets used during simulations, where the environment used our graphic animation utility described in Sect. 4; another set of subnets, used during the analysis phase; finally, a third set of subnets was used in case of automatic code generation, where the environment used serial port communication.

Figure 3 shows the subnet of the Central unit. The work group identified the five following processes: Input event handler receives changed input (from PIDs) and determines whether alarms are to be generated. Time handler controls time dependent data and handles time-out events. Control panel handler takes care of the user interaction to/from the control panels by sending menus to the control panel and executing user commands. Log handler logs events to the printers and the internal logs, e.g. when a detector triggers an

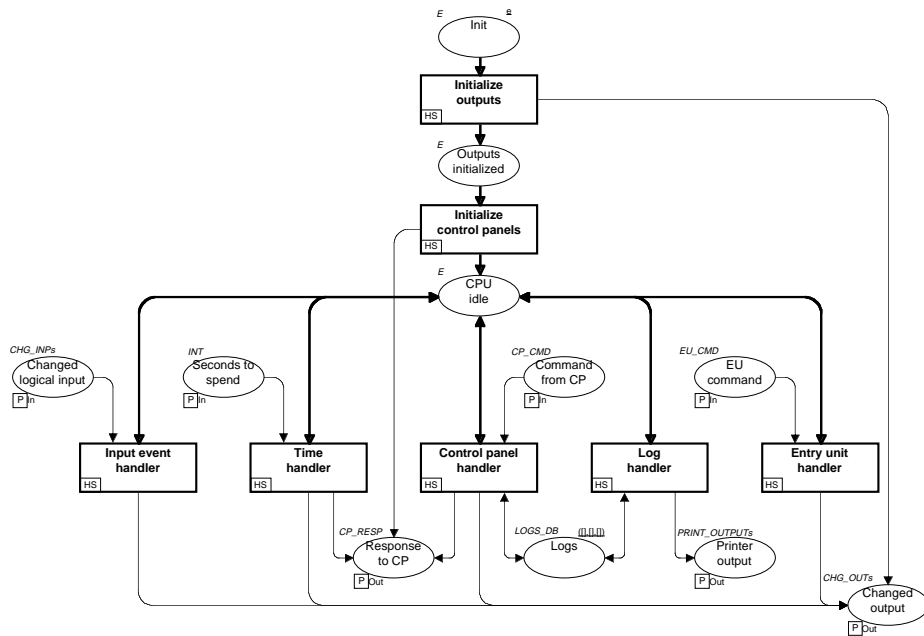


Fig. 3. Central unit

alarm, its name and alarm type will be logged. Entry unit handler handles entry unit commands, e.g. set/unset requests and user-code ratification. The processes can be activated if there are tokens on their input ports, representing environment changes, e.g. commands entered by the user on the control panel.

Central unit starts by initialising the outputs (e.g. switching on the control panel power-light) and sending start-menus to the control panel. It can then execute *one* of the five processes. The CPU idle place ensures that no more than one of the processes is chosen and finished before another can proceed. This is done in order to prevent the processes from operating on shared data simultaneously, which might lead to inconsistencies. Moreover, it reflects that only one processor exists in the hardware which is to be used for implementation.

Figure 4 shows the subnet of the Input event handler, describing its intrinsic behaviour. The sequential process-flow is indicated by thick arcs. White arrow-heads are used on one (the less important) of the two arcs that are needed in order to update a list. The %-operator used in the arc expressions returns the value to the right if the list of boolean expressions to the left all evaluate to true, but if one of the boolean expressions evaluates to false an empty multi-set is returned.

On the leftmost top place, changed input is given by the Environment (see Fig. 2), e.g. representing an opened infrared detector. An input can be either closed, opened, or sabotaged. The Receive changed inputs transition updates the input state on the Logical inputs fusion place. Subsequently, the Get

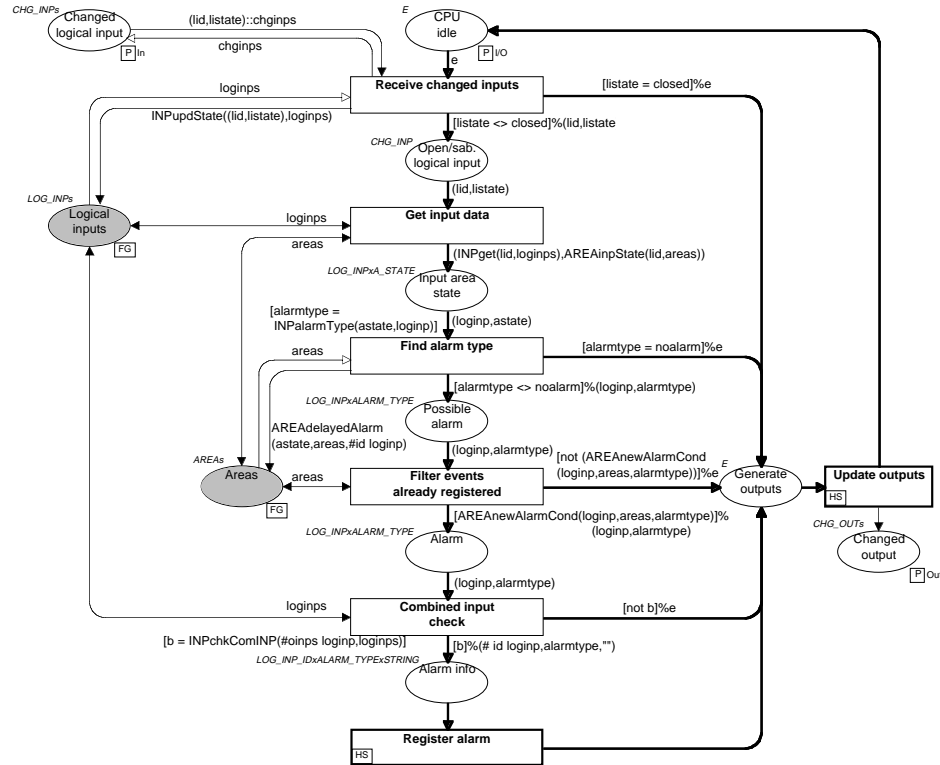


Fig. 4. Input event handler

input data transition retrieves the state of the area in which the input is situated. Find alarm type determines what alarm type is to be generated, e.g. a glass-break detector will generate an intrusion alarm type. Additionally, Find alarm type delays input changes belonging to areas temporarily unset by entry units. If an unset-command is not received from a control panel within 45 seconds, the delayed inputs will activate alarms (this is controlled by the time handler). Input not causing alarms will be filtered out. Then Filter events already registered ignores alarm conditions already registered. In order to reduce the number of false alarms input may be combined to depend on other input, which must all be open before triggering an alarm – this is controlled by Combined input check. Finally, an alarm condition is registered by the Register alarm substitution transition. The input event handler always ends by “calling” the rightmost Update outputs substitution transition, which re-evaluates the output expressions, as they might depend on the newly detected input change. A user-defined output expression is attached to each output in order to determine whether it is to be on or off. For example, the output expression for the horn could be: ALARM_COND(1) OR ALARM_COND(2), meaning that the horn is activated if there is an alarm condition either in area one or two.

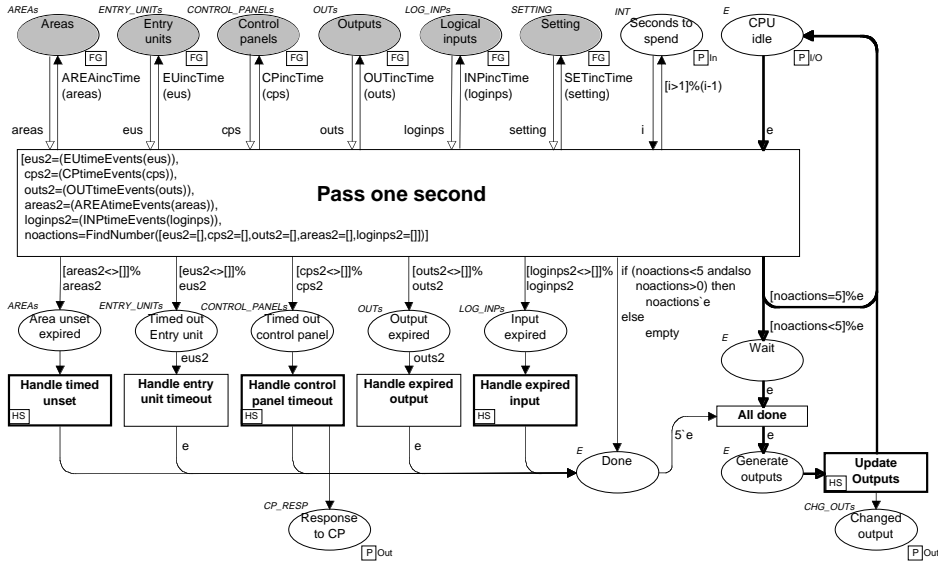


Fig. 5. Time handler

The Time handler manages all the time dependent data-structures, as shown in Fig. 5. All six fusion places have relative integer attributes, denoting the number of seconds before a time period expires. As an example, each area on place Areas has an integer-field which denotes the remaining seconds the area stays temporarily unset. The Pass one second transition increases time by updating the time attributes on all time dependent fusion places, and if any time-outs occur, they will be handled by the five Handle . . . transitions. For example, when a temporary unset period expires, the Handle timed unset subpage makes sure that the area-state is returned to set and that any delayed alarms are registered. After time-outs have been handled the Update Outputs substitution transition is called again, as on Fig. 4.

Timed CP-nets were not considered, as the CP-net was not intended for performance evaluations; instead, the CP-net should be able to explicitly cope with “real” clock-ticks from the environment. Therefore, the time has been integrated in the model by equipping the data structures with integer attributes, operating as local timers.

Having described two of the central unit processes, we remark that the user-interaction between the control panel and entry unit also constitutes a great portion of the model. All together, the model has 38 pages with 95 transitions and 325 places. In order to reuse identical net-structures, seven subpages have between two and four instances (copies). Furthermore, the model makes use of approximately 4,000 lines of CPN ML declarations and functions. CPN ML is the inscription language of Design/CPN and is based on the functional language Standard ML (SML) described in, e.g. [U1194].

4 Simulations

Simulation is an important instrument for debugging and verification of CP-nets. It gives the developer an improved understanding of the dynamics of the model.

In order to enhance the user-interface during simulations, we constructed a graphic animation utility. This utility supports two-way communication between the user and the simulator. The CP-net animates graphic objects while the user can feed the CP-net with input, based on selections. Hereby, a more user-friendly simulation is obtained. In fact, the user does not have to inspect the CP-net, but can concentrate on how the net responds graphically to his actions. This is particularly useful when presenting the CP-net to users without any knowledge of CP-nets. But it also became useful when testing the CP-net in the simulator. The animation utility allows the user to build his own graphic "scenes" composed by objects such as: buttons, arcs, bitmap objects, all of which can be manipulated during simulation to mimic/imitate their real-world counterparts. In this way, the user can visualise the CP-net's response to different scenarios and display interesting system-states.

We have constructed a library for Design/CPN using the built-in routines for manipulation of graphic objects [OAF93]. It is called *Mimic/CPN* [RS95a] and its main primitives are to: hide or show objects, move or align objects, hide or show connectors, select an object/connector by clicking on it with the mouse, combine multiple user-selections, and save and restore states of graphic objects. All library functions are to be called from transition code segments, which are pieces of SML code, executed whenever the corresponding transition occurs. The graphic objects are built within the editor, and their graphic appearance is defined by the user. This allows the user to customise the graphic layout for his own specific needs.

The "mimic" name is actually inherited from the security system, in which an optional *mimic-board* is included, showing a drawing of the covered building with lights mounted underneath. The lights indicate various system-states such as: area states, input states, alarms. In order to model the mimic-board, we use a scanned picture of a building with detectors represented as circles, output (e.g. indicators) as squares, and icons for various devices: clock, horn, entry units, control panels, and printers (see Fig. 6). When simulating, the environment subnet updates the graphic states of light-output, which are shown as either filled or empty squares – depending on whether the state is on or off. If, for instance, an output changes to on, the old square will be hidden, and a previously hidden filled square appears at the same point. Horn and flash output are shown as icons. Printer text, sent by the central unit, is appended in text-boxes on separate pages; which opens when the log icons are clicked. All graphic updates are done from transition code segments by means of *Mimic/CPN* functions.

The animation utility also allows the user to select from graphic objects, thereby making it possible for the user to control simulations. For example, when making an input change the user clicks on a detector and then selects its new state from the closed/opened/sabotaged symbols on the mimic-board. We also made entry unit commands (set and unset) and user-codes on the code

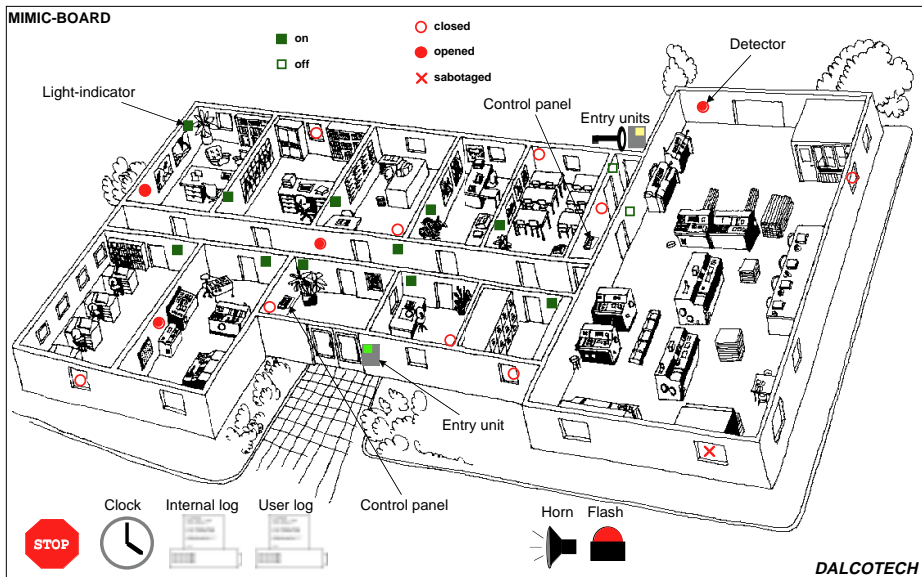


Fig. 6. Mimic-board

entry unit by letting the user click on the keypad – as he would do in the actual security system. When the user clicks the clock icon, a dialogue box will appear, and the entered amount of clock-ticks will be reported to the central unit. The entire control panel user-dialog, where the user browses in menus, received from the central unit, and selects commands has been implemented using the animation utility (see Fig. 7). It should be noted that the animation utility and the simulator are synchronised, i.e. when a select function is used, the simulator actually waits until an object has been chosen by the user.

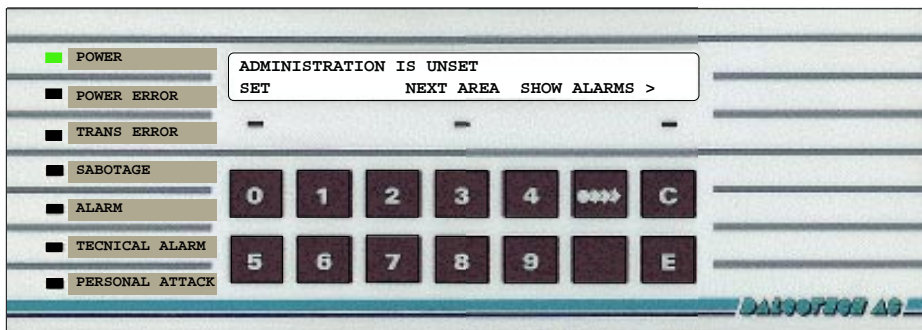


Fig. 7. Control panel

The work group used an iterative approach of modelling and simulation cy-

cles. The first prototype was gradually refined, and eventually, it constituted the final model. Simulation was used to test and investigate the net behaviour – after having augmented or changed the CP-net. The animation utility was particularly useful for attracting the user’s attention visually, if the graphic objects did not correspond to the user’s expectations. Typically, simulations of 300-400 steps were performed. In these trial-and-error experiments, the animation utility was used for guiding and monitoring the simulations.

5 Occurrence Graph Analysis

The CP-net is used as a specification of the final program. Therefore, finding an error in the model by means of simulation or analysis can inhibit an error in the final program. Test simulations of the CP-net revealed many flaws in the model and provided feedback for improving the model during the entire design phase. At the end of the design phase, we applied occurrence graph analysis (also known as reachability graph analysis and state space analysis) in order to locate as many errors as possible before the Dalcotech engineers started implementing the system. It should be stressed that at the same time test simulations were performed by the security system engineers. The simulation experiments also continued after we finished analysing the CP-net.

In this section, we briefly illustrate the changes made to the CP-net in order to generate occurrence graphs. Furthermore, we give examples of how we investigated dynamic properties of the CP-net, and how we located errors in the model. For the analysis of the CP-net we used a prototype of the Design/CPN Occurrence Graph Tool [CJ95] – we refer to this tool as the OG tool. The OG tool makes generation of full occurrence graphs⁴ (O-graphs) possible. After generating an O-graph, a set of standard queries can investigate dynamic CP-net properties, such as boundedness, liveness, etc. Additionally, queries which are customised by the user allow checking of system specific properties. The O-graphs of the OG tool are integrated with the simulator in the sense that it is possible to switch from a node in an O-graph to the state in the simulator which it represents – and vice versa.

Preparing the Model for Occurrence Graph Analysis

In advance, we had little idea about whether O-graph analysis would yield noteworthy results when used on the large security system model. The only thing that seemed certain was that plenty of reductions of the model and its configuration would be necessary. Some of the colour sets in the CP-net are records with between ten and twenty fields which either change dynamically during simulation or are determined by the configuration of the system. Even if “infinite” colour sets (e.g. integers) were reduced to small finite sets, the number of different reachable markings would be enormous. Hence, we did not expect to prove

⁴ A full occurrence graph has a node for each reachable marking and an arc for each occurring binding element.

the correctness of the CP-net in all kinds of different configurations. Instead, we wanted to look for errors in the model and get an increased understanding of the behaviour of the CP-net by examining its dynamic properties. In this way, the analysis would become an “extended simulation” which investigated all possible occurrence sequences of small examples. This would supplement normal simulations which investigate “few” occurrence sequences of larger examples.

With the purpose of analysing the CP-net by generating O-graphs, we replaced the environment subnet used for simulations (which applied the animation utility) with “test environments”. An example of such a test environment is displayed in Fig. 8. The environment page is a subpage of the top page from Fig. 2.

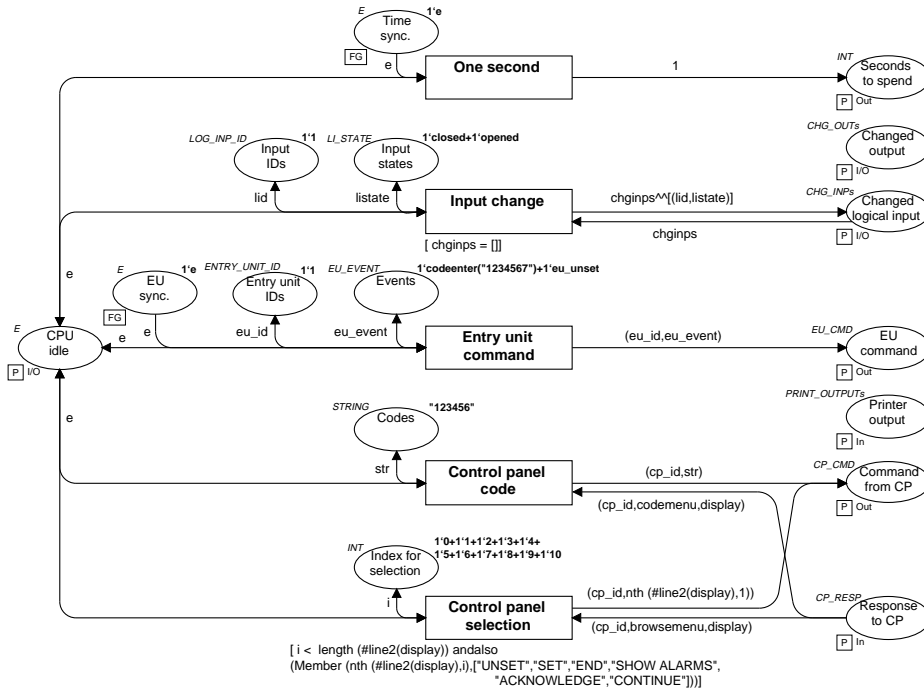


Fig. 8. Environment page for analysis

The displayed test environment produces clock-ticks (the One second transition), random input changes (Input change), random entry unit commands (Entry unit command), and random control panel commands (Control panel code and Control panel selection). The range of different input changes is decided by the tokens on the places Input IDs and Input states. Consequently, the displayed environment can only change the state of input number one to opened or closed. In similar ways, the range of different entry unit and control panel commands is controlled. However, when a browse menu is situated on Response to CP, a list of submenus can be chosen from. Here, a random submenu is chosen. The guard of Control panel selection ensures that only

“interesting” selections are made, i.e. selections which are necessary in order to test the important functions of the control panel.

In order to be able to fully generate O-graphs, we applied a minimal configuration with only one area, one input, one control panel, etc. Furthermore, some of the data fields of the elements have been modified. For instance, time-out periods are all set to expired after one second. Of course, this was a serious reduction of the security system. For instance, we were not able to examine conflicts between two control panels attempting to operate on the same area. However, the number of states exploded if, e.g. two areas and two inputs were included, and therefore, we focussed on testing the minimal configuration.

In addition to changing the environment and configuration of the security system we had to modify the model in order to reduce the state space. For example, we excluded the log-facility from the model, and we limited the list of changed output (on place `Changed output` of Fig. 8), so that it held no more than one element. Furthermore, we made the CPU `idle` place accessible from the environment page – thereby only allowing environment transitions to occur when the CPU was idle. The fusion places `Time sync.` and `EU sync.` were introduced for synchronising the transitions `One second` and `Entry unit command` with the central unit. For instance, the `One second` transition cannot send a second clock-tick until the time handler in the central unit has finished treating the first clock-tick. These modifications were not important for the behaviour of the model. However, they were necessary in order to obtain finite O-graphs of reasonable sizes.

In spite of all these reductions of the CP-net, fully generated O-graphs could only be obtained on small test examples. We could not achieve a fully generated O-graph for the environment displayed in Fig. 8 using the minimal configuration. Instead, we made O-graphs for selected parts of the system by, e.g. omitting entry unit commands from being generated in the environment. The largest O-graph generated consisted of 150,000 nodes and 250,000 arcs, for which the OG tool demanded 230 Mbytes of RAM. For practical reasons, we generally worked with smaller O-graphs which had up to 50,000 nodes.

Investigating Dynamic Properties of the CP-net

Having described the problems connected with constructing O-graphs from the CP-net we now look at the results we achieved from the reduced CP-net. Despite the reductions, investigations of the dynamic properties of the CP-net helped improve the confidence in the model. Furthermore, we located approximately 15 non-trivial errors, of which we will describe a few.

In order to check dynamic properties of the CP-net we made *queries* using the OG tool. A set of standard queries makes it possible to investigate the reachability, boundedness, liveness, fairness, and home properties corresponding to the propositions of Sect. 1.4 in [Jen94]. As an example, a single function-call returns a list of all transition instances which are live.

The test environment allows infinite occurrence sequences. As we were able to fully generate O-graphs, the infinite occurrence sequences must be *cyclic* in

some way. This made it interesting to investigate the set of strongly connected components (SCC) of the O-graph. Having generated an O-graph, the OG tool allows generation of an SCC-graph which has a node for each strongly connected component and thereby essentially identifies the cyclic structures of the O-graph. A terminal component of an SCC-graph represents a set of nodes in the O-graph from which no simulation can escape (while a terminal node of an O-graph represents a dead marking). When using test environments as shown in Fig. 8, we found that only one terminal strongly connected component existed. Thus, the individual nodes of the terminal component are home markings. This agrees with the intentions for the security system model. If alarms are generated, they can be acknowledged (and thereby removed), and if an area is unset, it can be set again, and so forth.

In general, the standard queries increased our confidence in the model. For instance, we established bounds on places and determined which transitions were live. However, some properties could not be verified by means of standard queries. For this purpose, the OG tool provides functions for traversing nodes, arcs, or strongly connected components of the O-graph, using user-defined functions to get the desired information. As an example, we made a query which searched all nodes of the O-graph and checked that only one process token existed.

The most important property of the security system is to report an alarm to the alarm receiving centre whenever a detector is triggered under illegal conditions. We applied the functions of the OG tool to make queries that found all nodes where “illegal” input change had arrived at the `Open/Sab. Logical input` place of Fig. 4. For all these nodes, we found the first successor node in which the input change had been treated by the input event handler. On this set of nodes in which the input change had been treated, we verified that the system was in an alarm condition and that the horn had been turned on. Thereby, we had proved that in a minimal configuration, illegal input changes to the security system always causes alarms to be generated.

As described above, we investigated standard as well as system-specific dynamic properties of the security system model. However, we also found a series of errors in the model by means of O-graph analysis. The functions used in the model all raise SML exceptions if something goes wrong. If an exception is encountered during O-graph generation, the generation halts. The OG tool makes it possible to switch a node in the O-graph into its state in the simulator. In this way, the marking which causes problems can be examined in the simulator. Furthermore, the OG tool makes it easy to find predecessors (and successors) of nodes in the O-graph. By examining predecessors, an occurrence sequence leading to the critical node can be found.

As an example, we discovered that the O-graph generation halted when the control panel timed out (which normally happens after 30 seconds of idle time). The error appeared when a user had been acknowledging alarms, a time-out had occurred, and the user had chosen to acknowledge alarms again. It turned out to be caused by an index error in the control panel data-structure. It is unlikely, that the occurrence sequence leading to the error would have been used

– revealing the error – in further test simulations. Therefore, the advantage of calculating all occurrence sequences in the O-graph was valuable in this case.

The OG tool makes it possible to display selected parts of the marking of nodes and the binding element of arcs. By examining these markings and binding elements, we also found a number of errors. For instance, we discovered that the list of area-data situated on the Areas place (see Fig. 2) was expanded (its elements were duplicated) during certain occurrence sequences. As the configuration of the system ought not to change during execution, this was an obvious error in the model. However, we had not encountered the problem during the previous test simulations.

The main drawback of making full O-graphs for the complex CP-net is that only greatly reduced configurations can be used if a state space explosion is to be avoided. We *did* make some O-graphs for larger configurations (with, e.g. two control panels) and a wider range of test environments. However, in order to reduce the state space we had to execute the central unit only a fixed number of times (normally one time). Thereby, the O-graphs became totally acyclic, and most of the standard dynamic properties of the O-graphs became uninteresting. For instance, no transitions were live, as no cycles existed in the O-graph. However, we also found errors in the model by this approach. As an example, we discovered that inconsistencies could occur when users at separate control panels were acknowledging alarms. A user could attempt to acknowledge an alarm which was already acknowledged by another user. As the error occurred when two users made certain selections at the same time from two different control panels, it is uncertain whether the error would have been located by further test simulations (which cannot test all such combinations).

All in all, O-graph analysis turned out to be a very efficient way of debugging the reduced CP-net. We found approximately 15 non-trivial errors in the model by this approach. Some of these errors would have existed in the final implementation of the security system, if they had not been located by O-graph analysis. Furthermore, investigations of the dynamic properties improved our understanding of the model behaviour. For these reasons, preparing the model for O-graph analysis proved worthwhile despite the difficulties it involved. Consequently, we can recommend using O-graphs for analysing CP-nets – even if the CP-nets are complex and only parts of them can be examined.

6 Evaluation

As the design of the system evolved throughout the modelling process, the work group frequently had to reject or radically change parts of the model. The most serious changes (apart from rejecting the client/server approach) were made in May '95 after employees from Dalcotech had learned more about the requirements that a security system needs to fulfil in order to be approved by the German standard for security systems. Furthermore, monthly meetings with the consultants induced a number of changes to the model. In this way, the modelling process was conducted in a *prototyping* fashion.

In parallel with constructing the CP-net, a written requirements specification was worked out. For those of us without knowledge of security systems, this was an important aid in the modelling process, as we could see a detailed description of what we were supposed to model. The detailed CP-net design made it possible to elaborate the requirements specification. The correlation between the requirements specification and the model was useful and also resulted in *two* system descriptions instead of one. This was useful later on, as both were consulted in order to retrieve information about the system.

In Fig. 9, an overview of the activities in the project is displayed. It illustrates how the requirements specification, the animation utility, and the implementation independent model were developed in parallel with the final model. The

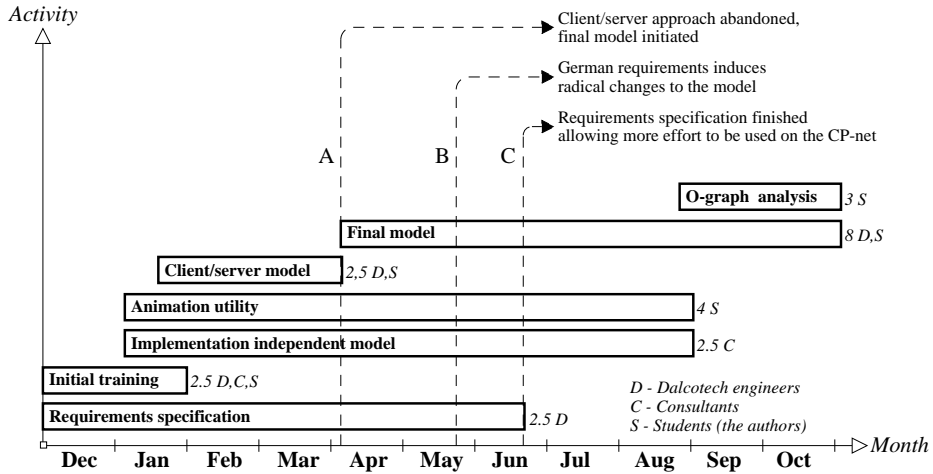


Fig. 9. Overview of project activities

number after each activity is an *estimate* on the amount of man months used on the activity. After each estimate, it is indicated who performed (the main part of) the respective activity. As an example, the figure illustrates that the Dalcotech engineers did not work on the implementation independent model, the animation utility, or the O-graph analysis. However, they did approximately half the work on the client/server model and the final model. The A, B, and C lines represent important events in the modelling process. Totally, two man years were used on the displayed activities. Fortunately, the development of the animation utility is only needed once, and in future CP-net projects at Dalcotech, the initial training of the employees is not necessary.

Apart from the activities displayed in Fig. 9, investigations of the feasibility of automatic code generation were conducted by the consultants and Dalcotech employees throughout most of the project. The SML code of the simulation-engine was extracted for automatic code generation, and was isolated by ignoring, e.g. all code used for graphic updating. The SML code was then ported to the Moscow ML [RS95b] environment, which is based on the Caml Light [Ler95]

runtime system. The generated code is small in size, but the interpreted programs run about ten times slower than compiler generated code. The stand-alone executable of the SML code used approximately 450 Kbytes of RAM, and the data also took up 450 Kbytes on the 80386 PC-card which is used in the central units of the security system. The results of these tests were promising, as it turned out to be possible to execute the SML code in its new environment. However, the automatically generated code was too slow and optimisations of the code were needed, if it was to be used in the final system. At this point Dalcotech decided not to proceed with automatic code generation. A tight schedule, lack of resources, and uncertainty about the success of this new approach were the main reasons for this decision.

The implementation of the security system began in October '95 after having finally decided not to proceed with automatic code generation. The CPN ML functions and colour sets are translated into C/C++ functions and types. Furthermore, the structure of the CP-net is used as a template for the C/C++ program. In order to implement the CP-net, some changes of the design have been necessary. However, the Dalcotech employees introduce such changes by updating the CP-net before implementing the changes. Thereby, the CP-net is constantly up-to-date with the actual implementation of the system. It should be noted that no automatic verification tool is used for checking that the C/C++ program is consistent with the CP-net. It is of course possible that the changes to the CP-net and the C/C++ code in itself will contain new errors. With respect to the adjusted CP-net, it was considered to apply O-graph analysis again in order to locate such errors. However, due to lack of resources this was not done. Instead, systematic test runs are performed.

Using CP-nets for Designing the Security System

The use of CP-nets in the Dalcotech project proved successful for the following reasons:

1. CP-nets provided a graphic as well as a formal description of the system. It is valuable to have a formal model which describes the main components and actions of the system in a graphic way. It is easy to find out technical details about the systems behaviour by looking at the CP-net.
2. Being able to execute CP-nets is a very powerful feature. By means of simulations, the members of the work group gained confidence in the existing CP-net. Using the animation utility described in Sect. 4, simulations became realistic tests of the functionality of the system, even though the hardware etc. was not yet developed at the time. The interface to the simulations resembles that of the final system. Thereby, the security system model can be demonstrated to customers.
3. O-graph analysis of the CP-net was a valuable aid for debugging the model. Some of the located errors could have emerged in the implementation of the system. Even though the analysis was limited by the complexity of the model, the errors which were found justify spending resources on O-graph analysis.

Finding and correcting errors in the final system can be very expensive. Therefore, the analysis methods associated with CP-nets are very valuable.

4. CP-nets have computer tools supporting their drawing, simulation and analysis. In our case Design/CPN provided almost all the functionality that we wished for. The tool has good drawing facilities for making CP-nets and makes it easy to syntax check and simulate the CP-nets. We did not meet barriers with respect to what we could model by means of the tool. Using the OG tool, we verified many dynamic properties of the CP-net, and additionally, we were able to locate a series of errors in the CP-net.
5. The use of CP-nets has improved the quality of software development at Dalcotech. The Dalcotech employees recognised CP-nets as “a new way of thinking” which takes time to learn, but which is worth the effort. CP-nets were used in all phases: from the initial sketches of the system to the final technical model which was used for coding the actual implementation.
6. CP-nets were used successfully by Dalcotech in connection with two other projects on the Dalcotech employees’ own initiative. Additionally, CP-nets are used for designing the low-level system which underlies the model described in this paper. Having become familiar with CP-nets, the security system engineers chose CP-nets for a rapid design of the low-level system. The security system engineers expect to use CP-nets in future projects, too.

The project also revealed problems with respect to using CP-nets in design of industrial systems:

1. CP-nets were unknown to the security system company as they are to most industrial companies. Consequently, training of the security system engineers was necessary. In spite of an intensive training course, it took some time before the engineers learned how to transfer their ideas into applicable CP-nets. Dalcotech would not have introduced CP-nets without the financial support from ESSI which covered expenses in connection with training, consultants, CP-net tool etc.
2. Using Design/CPN, it took a while to check the syntax, simulate etc. when the model began to grow larger. In general, it became somewhat cumbersome to add to the model and simulate the extensions. The Dalcotech employees also meant that better editing/debugging facilities of SML code in Design/CPN could have been very useful. A new version of the tool is presently being developed. The simulator is many times faster [HH94] and a new approach for declaring colours, etc. is included. Thereby, the problems concerning turn-around time are greatly reduced.
3. In order to apply O-graph analysis to the large CP-net many reductions of the model and its configuration were necessary. Therefore, analysing the CP-net was not a straightforward task. In some cases, using O-graphs with symmetrical or equivalent markings would decrease the need of prior reductions to the CP-net. However, the security system model is an example of a CP-net which does not have many symmetrical markings and which therefore is not well-suited for such an approach.

Having discussed the positive and negative aspects of using CP-nets for designing the security system, we conclude that CP-nets were very applicable in this industrial design project. The project demonstrates the value of the interactive simulations and formal analysis methods associated with CP-nets. The employees at Dalcotech recognised CP-nets as a useful design instrument and will continue applying CP-nets in future projects.

Acknowledgements

We would like to thank the Dalcotech company for letting us participate in the project. Furthermore, we thank Torben Andersen, Klaus L. Nielsen, Søren V. Hansen from Dalcotech and John Mølgaard from DELTA for fruitful cooperation and constructive ideas. Finally, we are grateful for the many valuable comments and ideas for this paper and our work in general from our advisors Kurt Jensen and Søren Christensen.

References

- [CJ95] Søren Christensen and Kurt Jensen. *Design/CPN Occurrence Graph Tool, User's Manual (vers. 1.0)*. Computer Science Department, Aarhus University, 1995.
- [CPN93] Meta Software Corporation, Cambridge, MA, USA. *Design/CPN Reference Manual for the Macintosh (vers. 2.0)*, 1993.
- [Dal94] Dalcotech A/S. *PRISMA C-91 System Manual*, 1994.
- [Dal95] Dalcotech A/S. *PRISMA C-96 Requirements Specification*, 1 edition, 1995.
- [HH94] Torben B. Haagh and Tommy R. Hansen. *Optimising a Coloured Petri Net Simulator*. Master's thesis, Computer Science Department, Aarhus University, 1994.
- [Jen92] Kurt Jensen. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use, Volume 1*. Monographs in Theoretical Computer Science. Springer-Verlag, 1992.
- [Jen94] Kurt Jensen. *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use, Volume 2*. Monographs in Theoretical Computer Science. Springer-Verlag, 1994.
- [Ler95] Xavier Leroy. *The Caml Light System, Release 0.7. Documentation and User's Manual*. INRIA, France. Available at ftp.inria.fr in directory lang/caml-light, 1995.
- [OAF93] Meta Software Corporation, Cambridge, MA, USA. *Design/CPN Internal Functions Programmer's Reference (vers. 2.0)*, 1993.
- [RS95a] Jens L. Rasmussen and Mejar Singh. *Mimic/CPN, A Graphical Animation Utility for Design/CPN (vers. 1.5)*. Computer Science Department, Aarhus University, 1995.
- [RS95b] Sergei Romanenko and Peter Sestoft. *Moscow ML Owner's Manual (vers. 1.31)*. Available at ftp.dina.kvl.dk in directory pub/mosml, 1995.
- [Ull94] Jeffrey D. Ullman. *Elements of ML Programming*. Prentice-Hall International Editions, 1994.

This article was processed using the L^AT_EX macro package with LLNCS style