KfK 4630
Oktober 1989

A Formal Net Specification of the Communication in a Distributed Computer System

H. Eggert
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE
Institut für Datenverarbeitung in der Technik

KfK 4630

A FORMAL NET SPECIFICATION OF THE COMMUNICATION IN A DISTRIBUTED COMPUTER SYSTEM

Horst Eggert

KERNFORSCHUNGSZENTRUM KARLSRUHE GmbH

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 7500 Karlsruhe 1

ISSN 0303-4003

Abstract

The *Petri Net Theory* provides a method for the *formal specification of* concurrent systems (e.g. distributed computer systems for diagnostic- and shutdown functions in Fast Reactors).

This method includes the *verification/validation* of the net specification about the causal behavior of the modeled concurrent system.

Especially the *Predicate/Transition Nets* (most important class of *high level nets*) provide very compact net specifications. Additionally, a Predicate/Transition Net can be described by different levels of a system abstraction (for example in form of a *net morphism*).

So it is possible to describe a concurrent system in a formal manner and to validate its causal behavior already during the design phase in the software live cycle. For example it can be shown, that all global states of an application protocol do not include *deadlocks* and *livelocks*.

Formal net specifications and the results of their analysis raise the *reliability* of software systems themselves and also the quality of the software documentation.

For the verification and documentation of net specifications *computer tools* are an absolute necessity.
In our case the operating system UNIX* is available with a special *Petri Net Tool* **PROVER** (**PR**edicate/Transition Net **O**riented **VER**ification System).

In this environment a net specification was developed which provides a formal description of an *application protocol* for a communication system. This communication system is the central part of our distributed *Fast Reactor Diagnostic System* **DESYRE** (**D**iagnostic **E**xpert **SY**stem for **RE**actor Surveillance).

Zusammenfassung

Eine formale Netz Spezifikation zur Beschreibung der Kommunikation in einem verteilten Rechnersystem

Die Petri Netz Theorie beschreibt eine Methode zur formalen Spezifikation nebenläufiger Systeme (z.B. verteilte Rechnersysteme zur Diagnose und zur Abschaltung in Schnellen Brutreaktoren).

Diese Methode enthält Möglichkeiten zur Verifikation/Validation der Netzspezifikation in bezug auf das kausale Verhalten des modellierten nebenläufigen Systems. Insbesondere die Prädikat/Transitions Netze (eine sehr wichtige Klasse der höheren Netze) unterstützen eine sehr kompakte Netzspezifikation. Zusätzlich können Prädikat/Transitions Netze verschiedene Ebenen einer Systemabstraktion beschreiben (z.B. in Form eines Netz Morphismus).

Auf diese Weise ist es möglich, ein nebenläufiges System in einer formalen Art zu beschreiben und die Validierung bereits während der Entwurfsphase im Rahmen des Software Lebens Zyklus vorzunehmen. Man kann z.B. zeigen, daß alle globalen Zustände des Applikations Protokolls keine Verklemmungen (deadlocks, livelocks) enthalten.

Formale Netzspezifikationen sowie die Ergebnisse ihrer Analyse erhöhen die Zuverlässigkeit von Software Systemen und ebenfalls die Qualität der Softwaredokumentation. Für die Verifikation und Dokumentation von Netzspezifikationen sind rechnergestützte Werkzeuge unbedingt notwendig.

In unserem Fall ist das Petri Netz Werkzeug PROVER (PRedicate/Transition Net Oriented VERification System) unter dem Betriebssystem UNIX verfügbar.

In dieser Umgebung wurde eine Netzspezifikation erstellt, welche ein Applikationsprotokoll für ein Kommunikationssystem beschreibt. Dieses Kommunikationssystem ist ein zentraler Teil des verteilten Diagnosesystems am Schnellen Brüter DESYRE (Diagnostic Expert SYstem for REactor Surveillance).

* UNIX is a trademark of AT&T

Inhaltsverzeichnis Page

1. Introduction

The *Fast Reactor Diagnostic System* **DESYRE** (**D**iagnostic **E**xpert **SY**stem for **RE**actor Surveillance) [1], [2], [3] ,[4] is realized as a distributed on-line computer system.

The system includes a set of special *detection computers* (e.g. for the detection of acoustic noise and temperature noise.)
These different detection systems use intelligent signal processing techniques to find out special faults and anomalies and to make partial diagnoses in an early state.

The system also includes a *process computer* for immediate processing of raw data and an on-line simulator for the simulation of faults.

Furthermore, the system contains a special *LISP computer* with an *expert system* for the inference of the correlated predications of the detection computers and the process computer.

A test system [5] allows for the description and the generation of test data in the computer network. Therefore, the inference of rules in the expert system can be easily tested with regards to their logical consistency by application of such a test system.

Figure 1 illustrates the computer network in form of an informal Petri Net (channel/instance net).

The communication between the different computers [6] within the seven levels of the *ISO/OSI communication model* [7] is realized with a *local area network (IEEE 802.3/Ethernet)* [8] and the *TCP/IP Protocol (Transmission Control Protocol/Internet Protocol)* [9].

Figure 2 illustrates the ISO/OSI - seven layer reference model in form of a channel instance net.

The level seven (*application protocol*) of the ISO/OSI communication model is realized in the following way:

- All above mentioned computers are used as *clients* in the computer network,
- an additional computer, the *data manager*, is used as a *server* in the computer network.

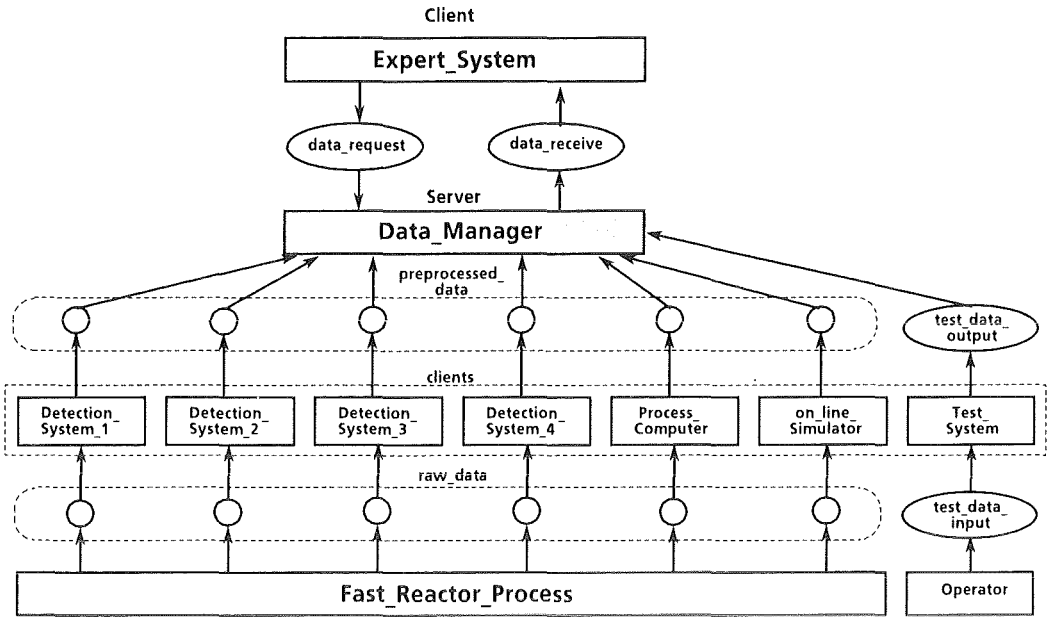


Figure 1: DESYRE: Distributed Expert SYstem for Reactor Surveilannce

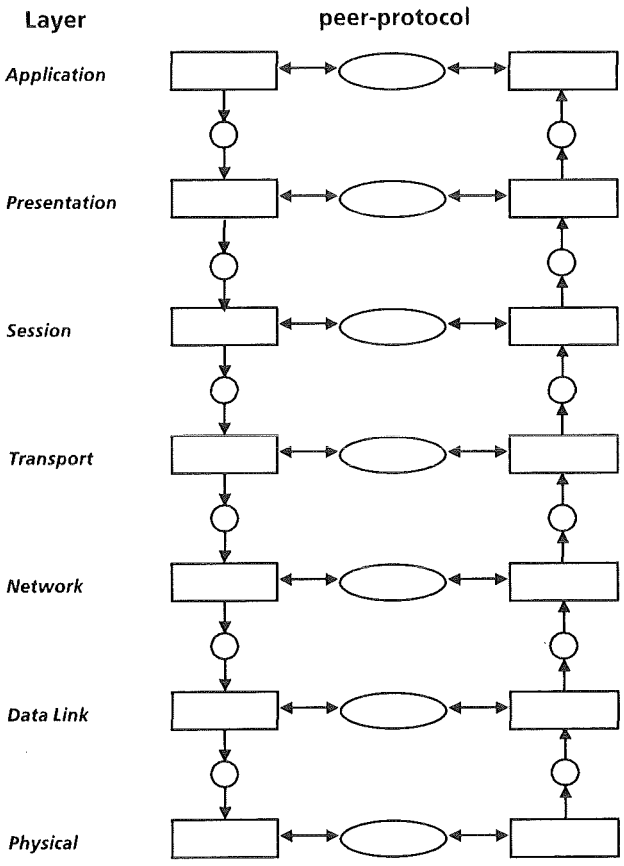


Figure 2: ISO/OSI - seven layer reference model and peer protocol

The data manager (server) *receives* preprocessed data from the detection computers and the process computer (clients) and *sends* these data to the expert system (client) (see also figure 1).

Therefore the data manager provides the following *abstract datatypes:*

- *FIFO* (First In, First Out)
- *RA* (Random Access) for history data.

The description of this application protocol is already made before an implementation. It is realized in form of a *formal net specification* with *Predicate/Transition Nets*.

Before presenting the method of Predicate/Transition Nets and the formal specification of the application protocol a general idea about the structure of the communication between clients and server shall be given.

Figure 3 represents this idea by an example of a File Transfer Protocol in form of a channel/ instance net.

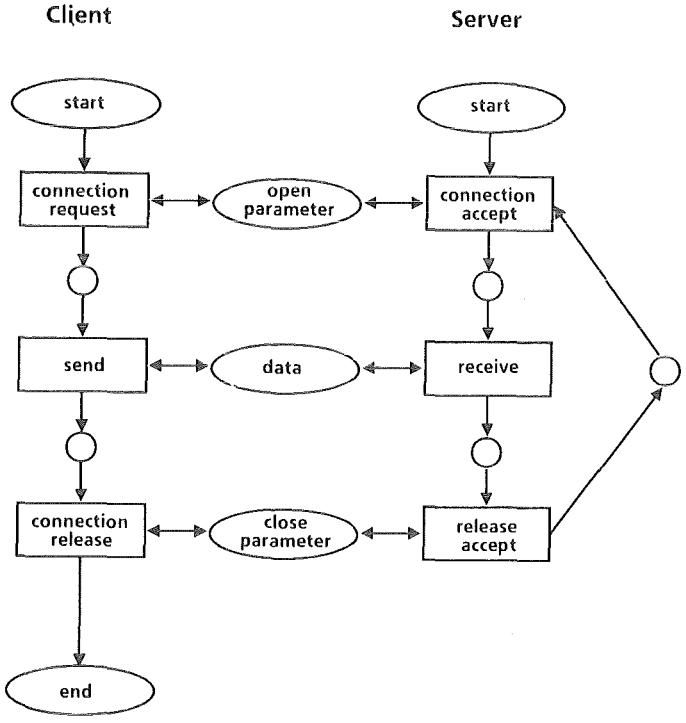


Figure 3: Application Layer: a short form of a File Transfer Protocol

Clients can send open parameters to the server trying in this way to get connections inside the local area network.

If a client gets an available connection, it sends or receives data to/from the server via this allocated connection. After this procedure, the active client gives back the connection to the server and the server can allocate this connection for other clients.

2. Formal specifications with Petri Nets

2.1 Preliminary remarks

The functionality of concurrent systems (e.g. application protocols inside a distributed computer system) may not completely be verified by tests as the number of dynamic states in the system is too large.

Therefore it is not possible to validate concurrent systems by only using test procedures.
Particularly in case of the integration of computer applications in technical environments with features of high safety (e.g. Fast Reactors) it is necessary to validate these computer applications with regard to the requirements of their reliability.

Verification of the most critical part of systems - *the concurrency* - is possible before realizing the implementation in form of a formal net specification.

Compilation of the net specification and simulation of the dynamic behavior of the net (all global states of the net) can be performed by application of a computer tool.

After the simulation all global states of the net are described in the *reachability graph*.

Another part of the computer tool allows for the interpretation of the reachability graph concerning deadlocks, livelocks and other failures of the design (for more information see chapter 3.2).

After validation of the formal net specification has been made the computerized automatic generation of the real runtime system is desirable. This however will not be discussed in this paper.

2.2 Place/Transition Nets

The Net Theory [10], [11] provides a method for the description of processes (the dynamic behavior of systems).
Here an axiomatic approach to the description of the information flow is made, assuming that the description of real processes is causal and does not include the arrangement into a time scheme.
Without any theory the figure 4 mediates an idea about *place/transition nets*.

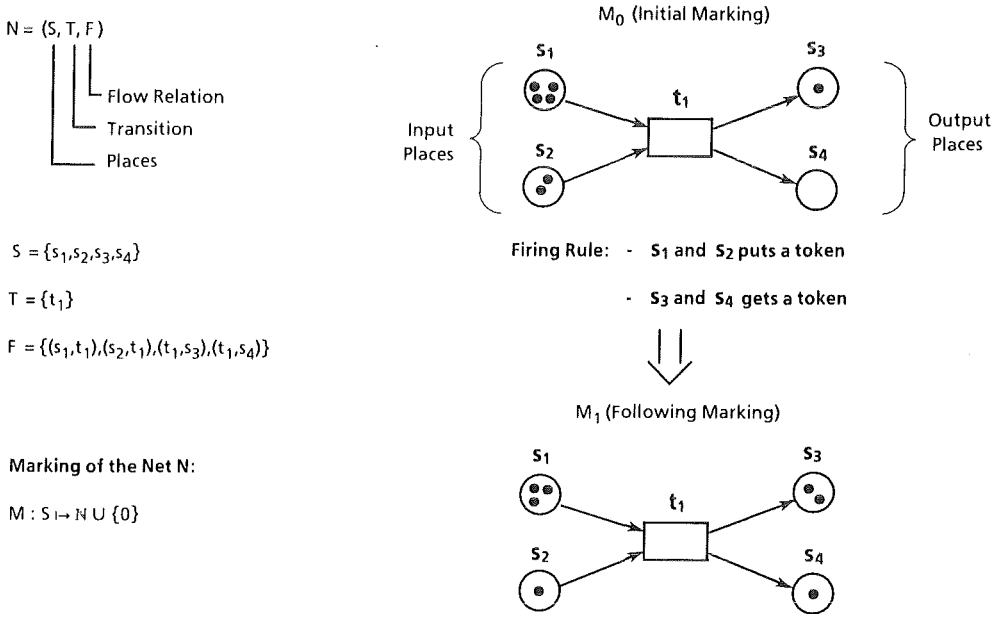


Figure 4: A Place/Transition Net

The static part of a net N includes *sets of places* (described as cycles in figure 4), *sets of transitions* (described as rectangles in figure 4) and a *flow relation* (described as arcs in figure 4).

We distinguish between input places and output places of a transition.

The dynamic part of a net N is described as *tokens* which means that the sets of places are mapped into the natural numbers.

The *firing rule* of place/transition nets implies that input places put a token and output places get a token. The first dynamic state in a net N is named *initial marking*.

A generalization of the firing rule is possible with the multiplicity of arcs and the capacity of places.

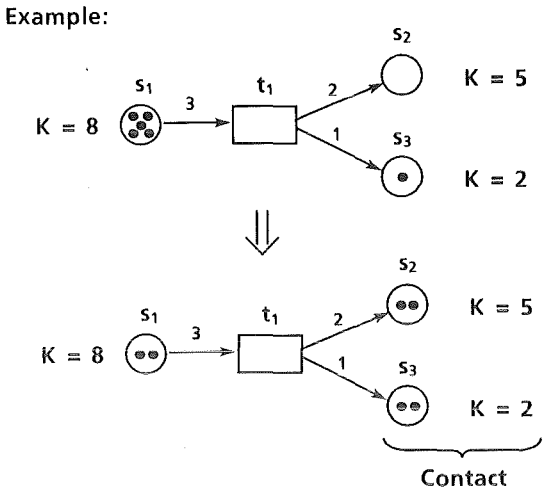
The multiplicity of arcs is defined as the "number" of tokens leaving the input places and arriving at the output places of a firing transition.

For the safety of nets the *capacity* of a place is necessary.

An example is given in figure 5.

Generalization of the Firing Rule

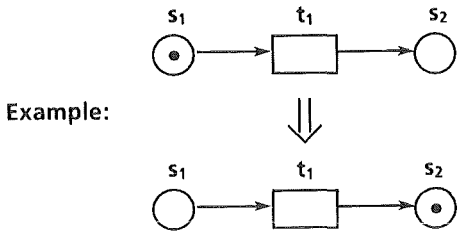
- Multiplicity of Arcs
- Capacity of Places (K)



An Upperbound of the Capacity of a Place
is necessary for the Safety of Nets

Figure 5: Generalization of the Firing Rule

Deadlock



Livelock

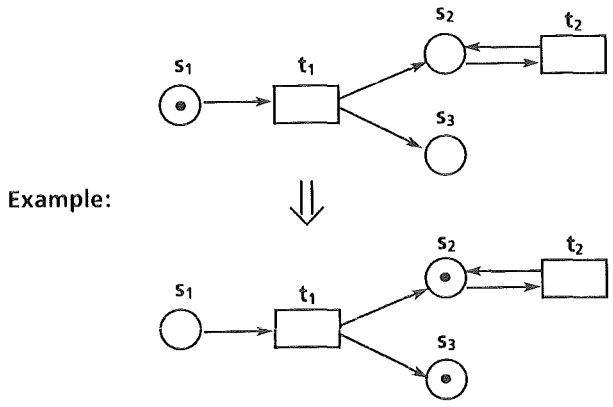


Figure 6: Deadlock and Livelock

Finally, a simple example of failures of the system design (see also chapter 2.1) will be shown.

Figure 6 illustrates a simple *deadlock* and a simple *livelock*.

Every real system working failure-free produces cyclical processes and not only one process step. This example describes the simpliest form of a deadlock.

Considering the example of a livelock the transition t_1 can make exactly one firing step and as a result the transition t_2 can make infinite firing steps. This demonstrates the simpliest form of a livelock.

2.3 Predicate/Transition Nets

For the description of extensive systems the place/transition nets discussed above are not sufficiently powerful because of the simple mapping of places into natural numbers and the simple firing rule in particular.

Much more powerful are the *predicate/transition* nets (high level petri nets) [12]. In the following these nets will be discussed without making any further remarks on the theory.

A partial formal overview is illustrated in figure 7.

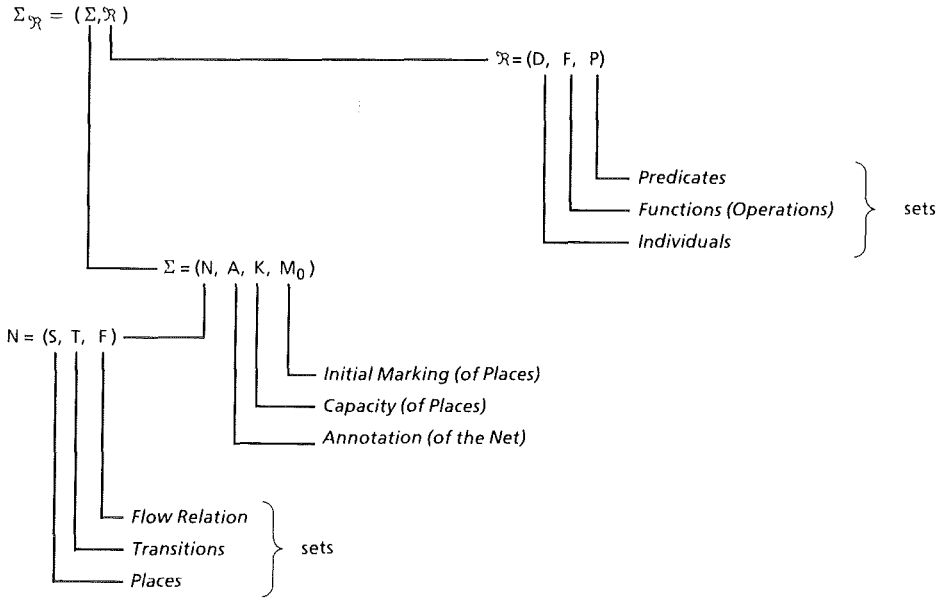


Figure 7: A partial Definition of Predicate / Transition Nets

Predicate/Transition nets consist of syntactical and semantical extensions of place/transition nets. The syntactical extension means the *annotation* of the net. The initial marking and the capacity has been already discussed in chapter 2.2.

The semantical extension implies the possibility to define *individual tokens* and their handling in the net. These individual tokens can be built from *sets of individuals*.

Furthermore, places can be mapped into predicates so that a semantical relationship exists between individual tokens, predicates and functions.

An example for the illustration of this abstraction is shown in figure 8 depicting the relationship between the formalism of predicate/transition nets which we already have discussed before and a simple application with only one place, one transition and the flow relation with two elements (two arcs).
This application describes the communication protocol on the highest possible level of the clients and the server inside the system DESYRE (see chapter 1).

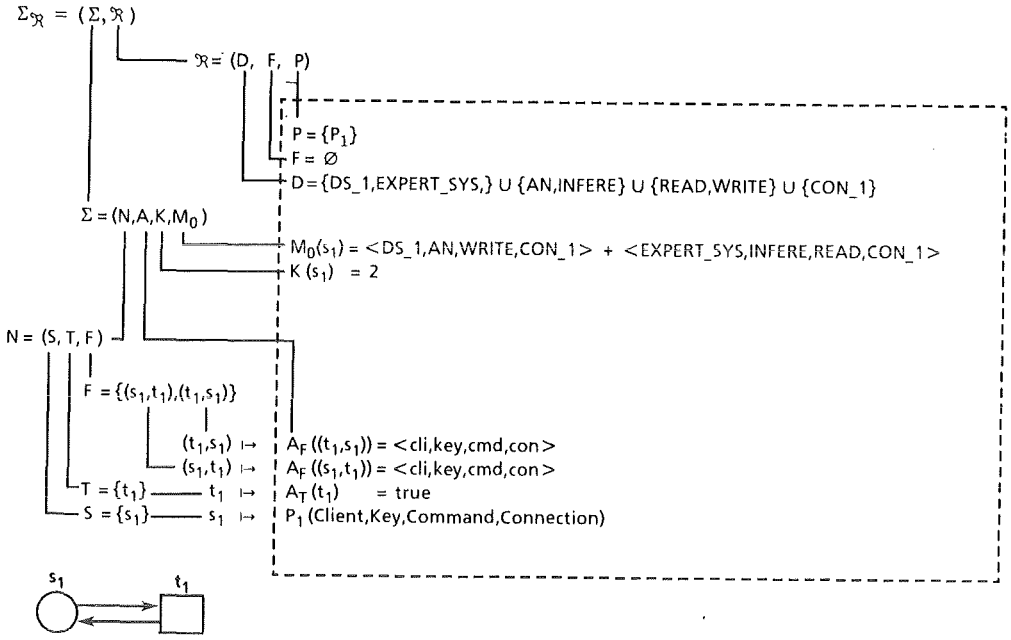


Figure 8: A Predicate / Transition Net in a Formal Form

The individuals in the union set D were defined as follows:

First subset of D:
- DS_1 (detection System 1)
- EXPERT_SYS (Expert System)

Second subset of D:
- AN (Acoustic Noise)
- INFERE (Inference)

Third subset of D:
- READ (receive)
- WRITE (transmit)

Fourth subset of D:
- CON_1 (Connection 1)

Based on these individuals we have built the two individual tokens <DS_1, AN, WRITE, CON 1> and <EXPERT_SYS, INFERE, READ, CON_1> which are used for the initial marking of the net in form of a formal sum.

Furthermore, the predicate P_1 of the place S_1 is defined.

P_1 has the following meaning:

Some *clients* (e.g. Detection System 1, Expert System) using *keys* (e.g. Acoustic Noise, Inference) and the *command* (READ or WRITE) receive or transmit from/to the server via the symbolic connection CON_1 inside the local area network.

In this example the set of functions is empty and the capacity of s_1 is 2.

The remaining will be easy to understand.

The flow relation (from the place to the transition and reverse) is described by the variables <cli, key, cmd, con>.
There is a direct association between the variables, the predicate P_1 and the individual tokens.

The annotation of the transition is true (logical formula) and does not include any further conditions in this case.

Figure 9 shows the same net representing the graphical elements of place/transition nets in a more visual form.

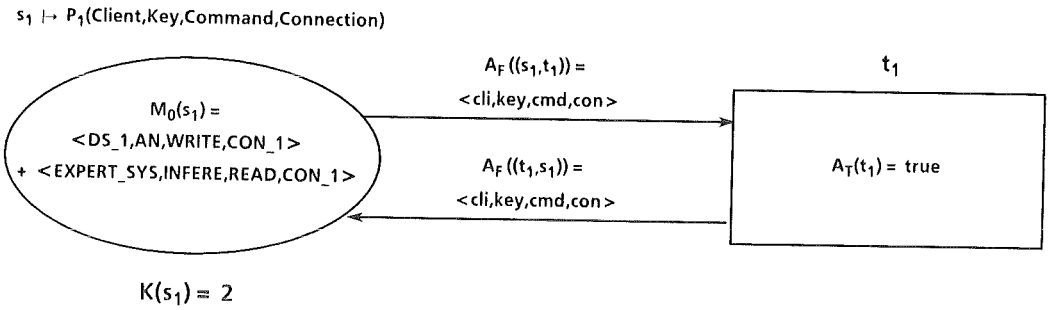


Figure 9: A Predicate / Transition Net in a Visual Form

The dynamic behavior of this net may as well be easily understood.

The reachability graph of the net which represents the dynamic behavior of the net includes only one node because all the following markings are exactly the same as the initial marking.

It is trivial that the node of the graph builds only one strongly connected component (SCC). That means that there is a mutual reachability of all markings in the net.

"Only one SCC" permits the interpretation that no deadlocks and no livelocks occurred and that a correction of the system design is not necessary.

On such a very high level of a model abstraction a view of single steps of the communication protocol of DESYRE is not possible and it should suffice here to understand the principles of predicate/transition nets.

In the next chapter we will try to give a more detailed view of the stepwise development of the communication protocol with predicate/transition nets.

3. A part of a formally specified communication protocol

3.1 Modeling by stepwise refinement

Designing a system in only one step is not possible. This would not be a good style of engineering and lead to unconditionally bad system designs which as a result produce unreliable systems.

The methods of structured programming [13] and stepwise refinement [14] teach us a good style of programming design and can be easily transferred into the context of the system design with nets.

The net specification of the application protocol consists two principles of *software engineering:*
- *stepwise refinement* of nets (net morphism),
- *embedding* of nets in other nets (net morphism).

Based on these principles a large net specification can be subdivided into a set of several consistent net parts.

The description of the net parts by different steps is more comfortable than the description of the whole net in only one step because the number of nodes and arcs in a single net part can be handled much easier.
Accordingly, the verification of the net parts in small steps is much easier than the verification of the whole net in only one step because the number of all dynamical states (*reachability graph*) in a single part is not so large.

In this way the description itself of the net specification and the verification of such a net specification (all causal behaviors of the whole application protocol) is possible.

Figure 10 illustrates a formal form of system engineering steps of the communication protocol of DESYRE.

Every node Σ in the graph stands for a symbolic name of a net and every arc ϕ in the graph is a symbol for a refinement step or an embedding step (a special refinement step) of a net.

Refinements and embeddings of nets are homomorphisms [15].

This means the following: An association between two objects a and b (in this case the nets a and b) is called morphism. If a and b have algebraic structures (predicate/transition nets include such a structure) and the association between the algebraic structures of a and b is compatible, then the morphism is called homomorphism.

Because of the large amount of steps and refinements only the nets Σ0 until Σ6 and Σa of figure 10 will be discussed here.
The dynamic aspects of the nets will not be considered now but will be discussed in chapter 3.2.

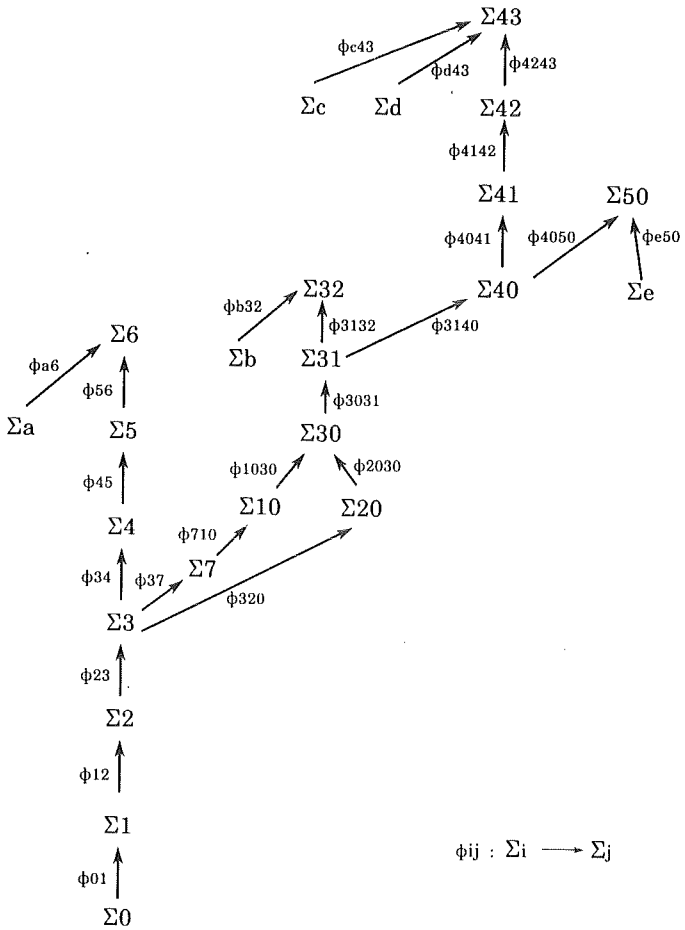


Figure 10: Graph of Stepwise Refinement and Embedding

Figure 11 illustrates the net Σ0 which has been discussed in chapter 2.3.

We already know the annotation of the flow relation by variables.
The place has the symbolic name client_message_connection, the transition has the symbolic name transmitter_receiver. We assume the annotation of the transition in the form true (in any other cases, if a transition has only the symbolic name and no further annotation, this assumption will be made).

The first step of refinement generates the net Σ1. This is illustrated in figure 12.

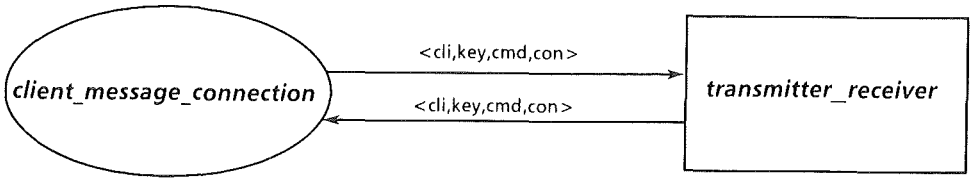


Figure 11: The net Σ0

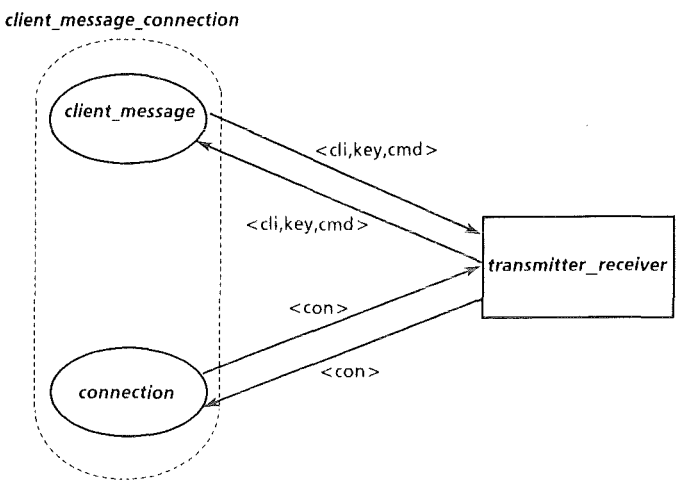


Figure 12: The net Σ1

We split the place client_message_connection into the two places client_message and connection. The flow relation is split as well, depending on the protocol semantic and the topology of the net. Another representation of the net Σ1 is shown in figure 13.

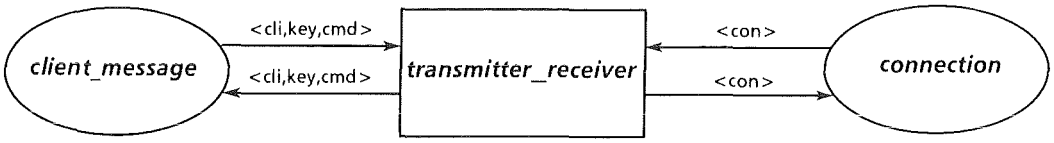


Figure 13: The net Σ1

Here exactly the same net is discussed but the overall idea of the client/server system gets more clear. The transition transmitter_receiver includes the activities of the server and on this level of model abstraction we can see that the server provides connections for the clients.

Now let us consider the next step of refinement which is presented in figure 14.

In the net Σ2 the place client_message is split into the two places client and message and the flow relation as well. This net provides more information about the clients showing the client as object and its message.

The next step of refinement is presented in figure 15.

The net Σ3 provides more detailed information about the transition transmitter_receiver because of its splitting into the two transitions open_accept and send_receive_close.

Clients get connections and leave connections. In which way the clients send and receive messages will not be discussed more concretely in this paper.

Let us consider the next step of refinement giving more information about the connection handling in the communication protocol.

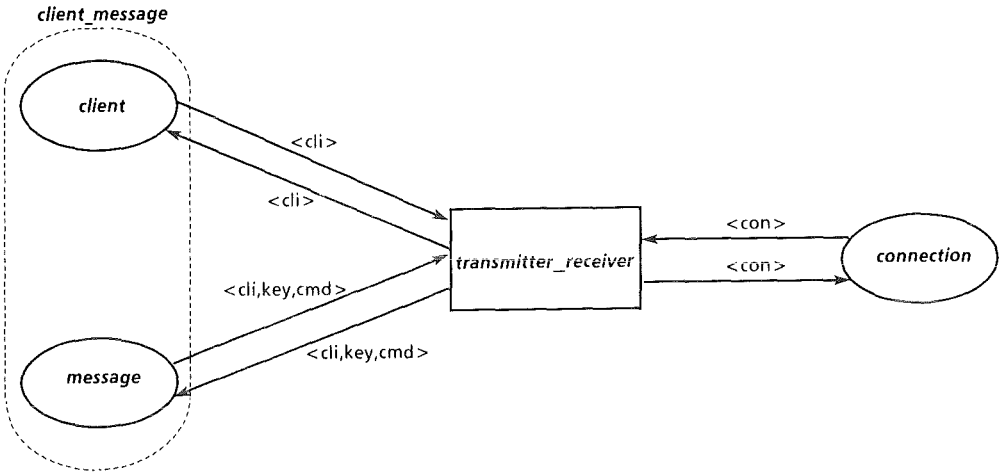


Figure 14: The net Σ2

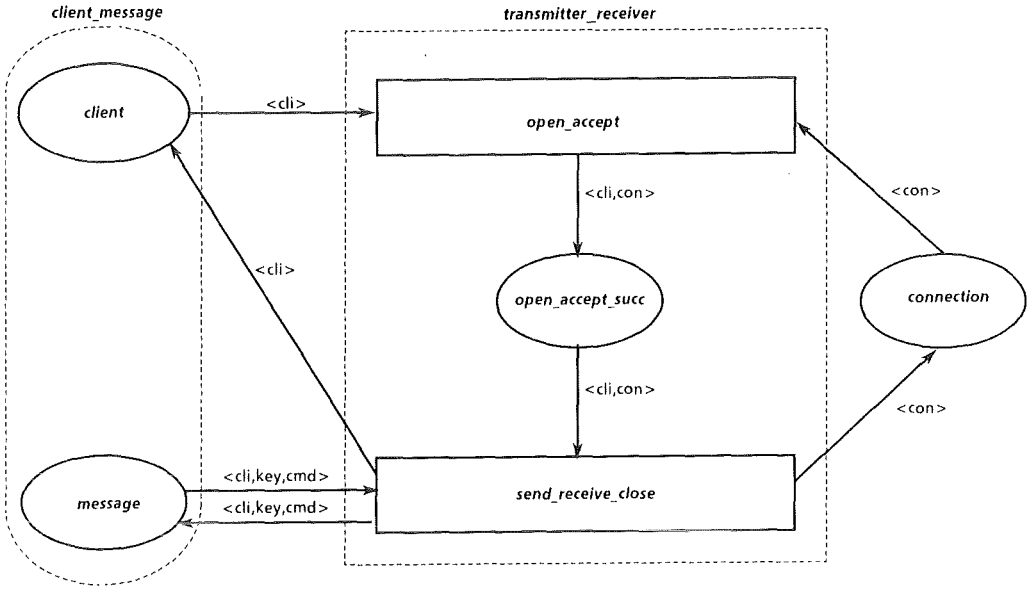


Figure 15: The net Σ3

Figure 16 presents additional information about the open of the client and the accept of the server.

The refinement step of the net Σ4 includes the refinement of the transition open_accept and the refinement of the place open_accept_succ in parallel.
This net clearly represents the separation of clients and server.

Figure 17 shows the next step of refinement by splitting the place accept_succ in the net Σ4 into the two places state_con and con_succ.

The refinement step in the net Σ5 is a suitable step for error handling if there is no connection available in the system.
If a client does not get a connection it cannot send or receive a message so that closing of this client would be of no purpose.

For a systematic interruption of the protocol procedure we define an additional transition error_empty_con with the condition if con=EMPTY_CON and, subsequently an embedding (a special step of refinement) of this transition in the net Σ5 will be designed.
The embedded net has the symbolic name Σa (see also figure 10) presented in Figure 18.

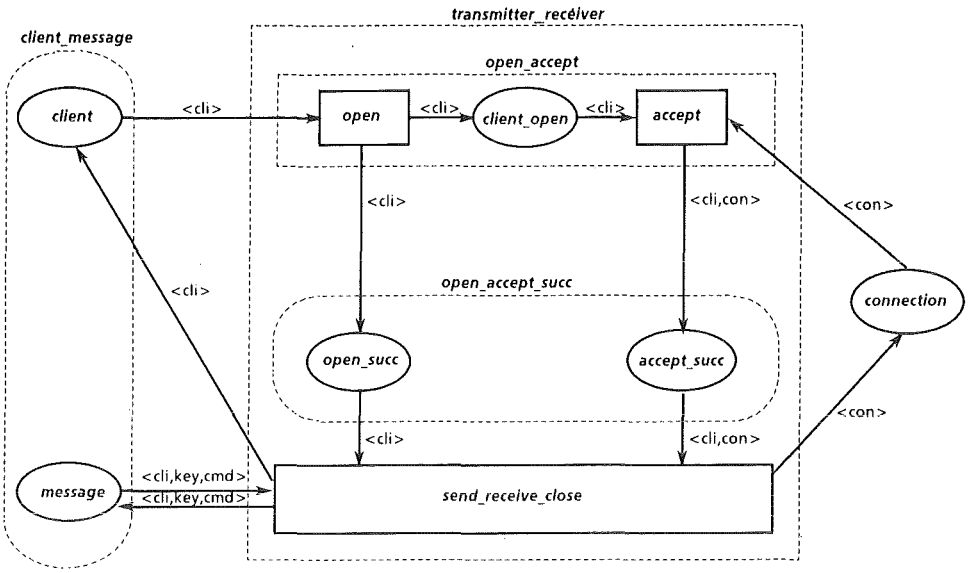


Figure 16: The net Σ4

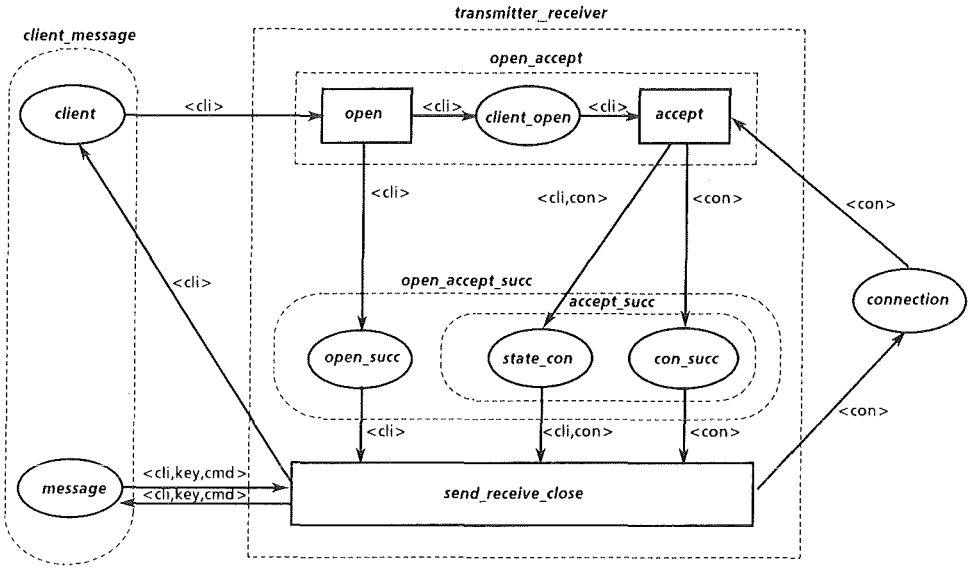


Figure 17: The net Σ5

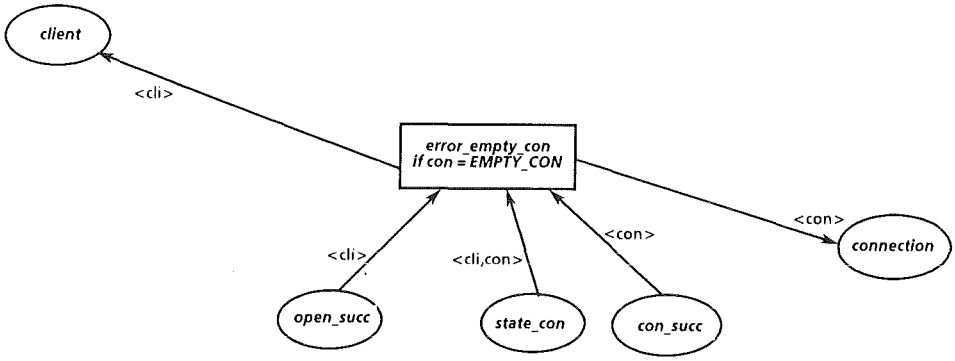


Figure 18: The net Σa

Figure 19 shows the net Σ6.
This net includes error handling of unavailable connections in the communication system.

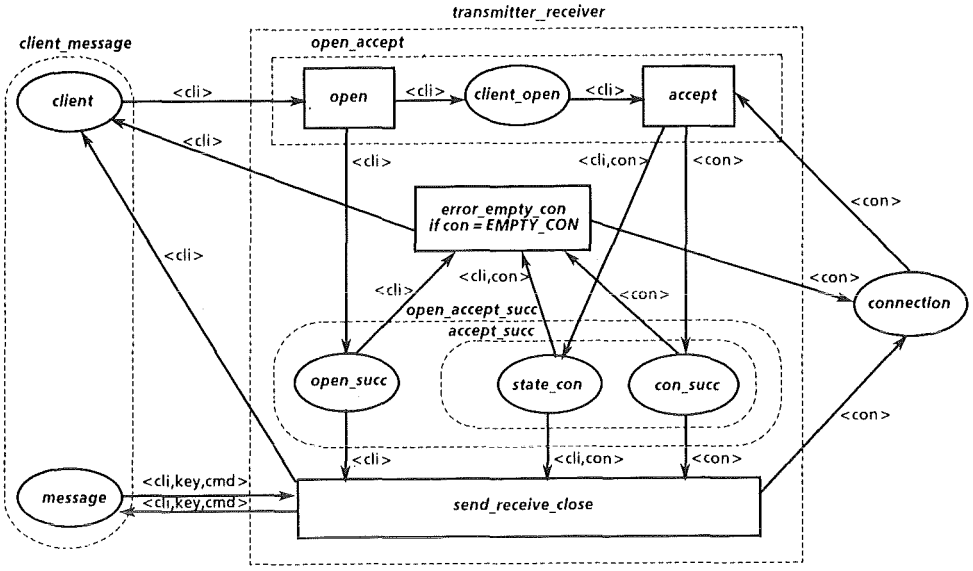


Figure 19: The net Σ6

At this point we will leave the engineering step of stepwise refinement and embedding. It should be enough for the suggestion of ideas about the modeling method with predicate/transition nets.

In the next chapter a presentation of a partial computerized verification of the net Σ6 will be given.

3.2 The verification of the net model

3.2.1 The Petri Net Tool PROVER

The actual version of the Petri Net tool **PROVER** (**PR**edicate/Transition Net **O**riented **VER**ification System) includes three operating modules [16].

Figure 20 illustrates the structure.

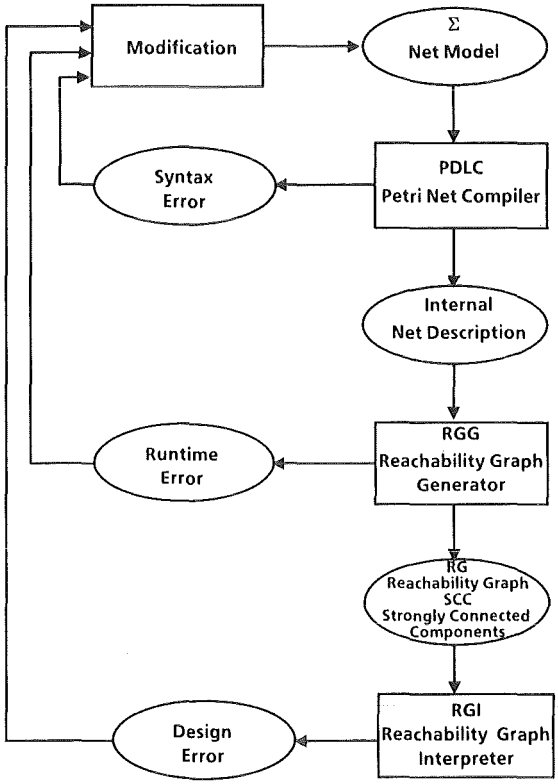


Figure 20: Internal Structure of the Petri Net Tool PROVER

(1) The Petri Net Compiler (PDLC) translates a net model which is described in the PDL (predicate/transition net description language). Syntactical errors and an internal net description will be generated.

(2) Based on the internal net description and the initial marking the Reachability Graph Generator (RGG) generates the complete reachability graph.

(3) The Reachability Graph Interpreter (RGS) applicates a number of different methods for the reachability analysis. The most important method is the analysis of deadlocks and livelocks.

For more information see [16].

3.2.2 The partial verification of the net model with PROVER

First of all the net model in the PDL has to be described. The PDL representation of the net model Σ6 (see figure 19) is demonstrated in the following.

```
petrinet        sigma_6;

const   max_cli         = 8;
        max_con         = 8;

var     cli:    (DS_1, DS_2, DS_3, DS_4, PROCESS_COMP, TEST_SYS, ON_LINE_SIM,
                 EXPERT_SYS);
        key:    (AN, DN, FT, CM, TN, TMFI, PRIM_PUMP, INFERE, EMPTY_KEY);
        cmd:    (READ, WRITE);
        con:    (CON_1, CON_2, CON_3, CON_4, CON_5, CON_6, CON_7, CON_8,
                 EMPTY_CON);

place   client (max_cli)                        'clients enters the system';
        message (max_cli)                       'messages of the clients';
        client_open (max_cli)                   'client send open';
        open_succ (max_cli)                     'open is successful';
        state_con (max_cli)                     'state of connections';
        con_succ (max_con)                      'connection successful';
        connection (max_con)                    'connections of the system';

trans   open                                    'open of the system';
        accept                                  'accept of the system';
        error_empty_con     if con = EMPTY_CON  'error no connection available';
        send_receive_close                      'send_receive_close of the system';

flow    open                    < client(<cli>)
                                > client_open(<cli>), open_succ(<cli>);

        accept                  < client_open(<cli>), connection(<con>)
                                > state_con(<cli,con>), con_succ(<con>);

        error_empty_con         < open_succ(<cli>), state_con(<cli,con>),
                                  con_succ(<con>)
                                > client(<cli>), connection(<con>);

        send_receive_close      < open_succ(<cli>), state_con(<cli,con>),
                                  con_succ(<con>), message(<cli,key,cmd>)
                                > message(<cli,key,cmd>), client(<cli>),
                                  connection(<con>);

mark    client:         <DS_1>+<EXPERT_SYS>;
        message:        <DS_1,AN,WRITE>+<EXPERT_SYS,INFERE, READ>;
        connection:     <CON_1>+<EMPTY_CON>;

endnet
```

Any further remarks on this PDL will not be interesting as the syntactical structure of this code is very simple (for more information see [16]) and the semantical structure of the net has already been discussed in chapter 3.1.

Just a remark on the initial marking of the net will be necessary.
The last part of the PDL representation of the net Σ6 is mark (that means the initial marking of the net Σ6).
In the place client there are two individual tokens in form of a formal sum,
<DS_1> + <EXPERT_SYS>.
The place message also includes two individual tokens in form of a formal sum,
<DS_1, AN, WRITE> + <EXPERT_SYS, INFERE, READ>.
The place connection includes two individual tokens in form of a formal sum,
<CON_1> + <EMPTY_CON>.
That means that the detection system 1 tries to write (transmit) the results of its detection method "acoustic noise" to the server via the symbolic connection CON_1.
And that means also that the expert system tries to read (receive) the results from the server via the symbolic connection CON_1, too.

These processes are concurrent on the shared resource CON_1.

The symbolic connection EMPTY_CON simulates the error handling (see chapter 3.1). In fact, much more concurrency and shared resources exist in the net model but we will not consider these features on this level of model abstraction.

Now let us call the net compiler PDLC with the following input:
pdlc sigma_6

The net compiler provides the following output:

```
PDL compilation summary : No errors reported

  Number of objects found, assorted by type:

       4 variables
      28 values
       0 functions
       7 places
       4 transitions
       0 facts

  in predicate/transition net SIGMA_6
```

After the compilation of the net Σ6 has been done we call the reachability graph generator RGG with the following input:
rgg -m sigma_6

The reachability graph generator provides the following output:

```
Reading PRT net model        ...done.
Building reachability graph. One dot = 12 markings.
. done.
Writing reachability graph   ...done.

Reachability graph has   14 nodes in    1 strongly connected components
```

By making the following input our dialog is transferred into the rgi subsystem:
rgi sigma_6

The rgi subsystem provides the following output and stops with the prompt rgi >.

```
Reading      reachability graph ...  done
Building strong component graph ...  done
Calculating transition dynamics ...  done
Calculating graph statistics    ...  done

rgi>
```

Now we are in the rgi subsystem and we select a subset of questions about the reachability graph of the net Σ6 with its initial marking which we discussed before.

```
rgi> show statistics

          Number of variables   :     4
          of values       :    28

          of places       :     7
            --"--  dead   :     0
          of transitions  :     4
            --"--  dead   :     0
          of overflows    :     0

          of facts        :     0
          of false facts  :     0

          of graph nodes  :    14
          of graph  arcs  :    38
          of SCCs         :     1

Mean rel. place density   %(places / marking)    : 74.5
Mean token      density    (tokens / place)      :  1.4
Mean attribute  density    (attributes / token)  :  1.6
```

As all the questions are answered positively by the reachability graph interpreter RGI it has been proved that the net Σ6 is free of any design failures.
For more information about design failures see [16].

```
rgi> show places

     Places :

     1. place      : CLIENT
        semantics  : clients enters the system
        arity      :     1
        capacity   :     8
        max. tokens:     2

     2. place      : MESSAGE
        semantics  : messages of the clients
        arity      :     3
        capacity   :     8
        max. tokens:     2

     3. place      : CLIENT_OPEN
        semantics  : client send open
        arity      :     1
        capacity   :     8
        max. tokens:     2

     4. place      : OPEN_SUCC
        semantics  : open is successful
        arity      :     1
        max. tokens:     2

     5. place      : STATE_CON
        semantics  : state of connections
        arity      :     2
        capacity   :     8
        max. tokens:     2

     6. place      : CON_SUCC
        semantics  : connection successful
        arity      :     1
        capacity   :     8
        max. tokens:     2

     7. place      : CONNECTION
        semantics  : connections of the system
        arity      :     1
        capacity   :     8
        max. tokens:     2
```

```
rgi> show variables

     Variables :

     1. variable : CLI
        range    : DS_1, DS_2, DS_3, DS_4, PROCESS_COMP, TEST_SYS,
                   ON_LINE_SIM, EXPERT_SYS
     2. variable : KEY
        range    : AN, DN, FT, CM, TN, TMFI, PRIM_PUMP, INFERE, EMPTY_KEY
     3. variable : CMD
        range    : READ, WRITE
     4. variable : CON
        range    : CON_1, CON_2, CON_3, CON_4, CON_5, CON_6, CON_7, CON_8,
                   EMPTY_CON
```

```
rgi> show transitions

    Transitions :

    1. transition   : OPEN (infinitely-firable, strongly-live & fair)
       semantics    : open of the system

    2. transition   : ACCEPT (infinitely-firable, strongly-live & fair)
       semantics    : accept of the system

    3. transition   : ERROR_EMPTY_CON (infinitely-firable, strongly-live & fair)
       semantics    : error no connection available

    4. transition   : SEND_RECEIVE_CLOSE (infinitely-firable, strongly-live & fair)
       semantics    : send_receive_close of the system
```

```
rgi> show deads

     The dead places : none.

     The dead transitions : none.
```

```
rgi> find livelocks

     The livelocks : none.
```

We leave the Reachability Graph Interpreter with the command quit.

```
rgi> quit
```

4. Conclusion

Formal net specifications and the results of their analysis raise the *reliability* of software systems themselves and also the *quality* of the *software documentation*.

In the future it may be necessary in West Germany to use formal net specifications and their verification/validation for *licence procedures* of the **TÜV** (Technischer Überwachungs **V**erein) in case of the integration of computer applications in the Fast Reactor and other technical environments with features of *high safety*.

Concerning the further development of the Petri Net tool PROVER we are currently working on the following components:

- graphical net editor with a better PDL representation and semantical features on homomorphism
- implementation of further possibilities of interpretation of the reachability graph.

We hope that for the future it will be possible to connect the Petri Net method with methods of algebraic specifications for algorithms.

This will be the basic assumption for the generation of real computer systems by formal system specifications.

References

[1] EGGERT, H., Ein Entwicklungskonzept für ein Störfall-Diagnosesystem, unpublished report, Kernforschungszentrum Karlsruhe, (1986)

[2] EGGERT, H., KELLNER, A., MASSIER, H., SCHADE, H.J., SCHLEISIEK, K., SCHRÖDER, R., Diagnosesystem zur Kernüberwachung von schnellen Brutreaktoren, unpublished report, Kernforschungszentrum Karlsruhe, (1986)

[3] EGGERT, H., SCHERER, K.P., KELBASSA, H.W., STILLER, P., DÜPMEIER, C. "Ein wissensbasiertes, verteiltes Rechnersystem zur Fehlerdiagnose am Schnellen Brüter, Informatik-Fachberichte Nr. 167, Springer Verlag, Berlin, Heidelberg, New York (1988), 749-761

[4] EGGERT, H., SCHERER, K.P., STILLER, P., "A knowledge-based on-line diagnostic system for the Fast Breeder Reactor KNK II", International Atomic Energy Agency, Specialists' Meeting on Advanced Control for Fast Reactors, Argonne, USA (1989)

[5] DÜPMEIER, C., EGGERT, H., Ein Testsystemprototyp unter UNIX für das Diagnosesystem am Schnellen Brüter, unpublished report, Kernforschungszentrum Karlsruhe, (1987)

[6] RUDOLPH, G., ROHNACHER, P., Realisierungskonzept für das Kommunikationsgesamtsystem, unpublished report, Kernforschungszentrum Karlsruhe, (1987)

[7] ISO 7498, OSI - Open Systems Interconnection - Basic Reference Model (1984)

[8] IEEE Project 802, Local Area Network Standards, Draft IEEE 802.3 - CSMA/CD Access Method and Physical Specifications, Revision D (1982)

[9] COMER, D. E., Internetworking With TCP/IP, Principles, Protocols, and Architecture, Prentice-Hall International, Inc., Englewood Cliffs, New Jersey,(1988)

[10] PETRI, C.A., "Concepts of Net Theory", Proc. Symp. on Math. Found, of Computer Science, High Tatras (1973)

[11] REISIG, W., Petri-Netze, Eine Einführung, Springer Verlag, Berlin, Heidelberg, New York (1982)

[12] GENRICH, H.J., LAUTENBACH, K., System Modelling with High-Level Petri Nets, Theoretical Computer Science 13, North Holland Publishing Company (1981) 109-136

[13] DAHL, O.-J., DIJKSTRA, E.W., HOARE, C.A.R., Structured Programming, Academic Press, London and New York (1972)

[14] WIRTH, N., Program Development by Stepwise Refinement, CACM, Vol. 14, No. 4, April 1971

[15] EGGERT, H., "Eine Anwendung von Petri-Netzen für eine partielle Prozeßbeschreibung und deren Abbildung auf Echtzeitelemente von PEARL", Vom Fachbereich Informatik der Technischen Universität Berlin genehmigte Dissertation, (1978)

[16] LESZAK, M., EGGERT, H. Petri-Netz-Methoden und -Werkzeuge, Hilfsmittel zur Entwurfsspezifikation und -validation von Rechensystemen, Informatik-Fachberichte Nr. 197, Springer Verlag, Berlin, Heidelberg, New York, 1989