This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Belli, F., Model-Based Construction and Implementation-Oriented Evaluation of Complex Systems, Proceedings of the IFIP Working Conference on Systems Engineering in Public Administration, Elsevier Science Publishers B.V., pp. 125-143

Model-Based Construction and Implementation-Oriented Evaluation of Complex Systems

Fevzi Belli[a]

[a]Electrical and Electronics Engineering Department, University of Paderborn, P.O. Box 1621, D-W 4790 Paderborn, Germany

Developing complex systems entails appropriate methods to be deployed in the iterative stages of construction and evaluation. This paper suggests to establish a solid model beforehand which then will be stepwise refined and enriched by procedural, declarative and heuristic knowledge of the specific domain. At the end of this process, the materialized system must be evaluated to validate the user requirements by means of meaningful tests and to predict the system's future behaviour by means of reliability determination. For the evaluation, we suggest to analyze the implemented system - not its specification, as recommended by conventional systems engineering - to generate the test procedures and test cases. Thus, we view the implemented system as its ultimate specification, because the original specification is usually obsolete when the implementation is carried out, and testing starts. To demonstrate the approach, we combine predicate/transition nets (also known as coloured/high-level Petri Nets of the General Net Theory) and Logic Programming for system modelling, implementation and evaluation.

1. INTRODUCTION

Our society increasingly deploys technical highly sophisticated tools endeavouring better services performed at cheaper prices. Also the public administration sector is expected to adjust and to keep pace with this situation. The result is the increasing complexity we have to master while developing and handling the systems we now have anywhere for any purposes.

We feel a system (or a process) *complex*, not only because it has many components and/or parameters which depend on each other. The systems we feel complex have additionally features, mostly being *pluralistic*, i.e. critical in the sense that they must not be ignored by severe sanctions; these features stem from needs for reliability, safety, integrity etc. Moreover, complex systems are embedded in other systems which are complex themselves.

In software engineering, our efforts to master complexity led to two different meanings (An excellent compendium on this subject is given in [13]):

- *Psychological Complexity.* As tentatively explained above, complexity can be viewed as an external feature of the system we experience when we try to understand it through observing its behavior and operational structure.

- *Computational Complexity* deals with the determination of the resources that algorithms need while being computed (if they are computable at all), e.g. time complex-

ity, space complexity. Computational complexity is closely related to the functioning of the system, therefore it can be viewed as an internal feature. This area is very intensively and extensively researched, constituting a well established and understood kernel field of computing science.

To master the complexity, we usually undertake two steps:

- *Hide it behind new notions nobody (yet) understands thoroughly!* Our first attempt to counter difficulties for understanding and handling a new situation is to identify it, i.e. take out the *old friends* we are familiar with and separate and name the new ones. Remember our architectural hierarchy *Data Processing*, *Information Processing*, and *Knowledge Processing* handling different problems and concepts of software engineering. Of course, we have many reasons for having made this distinction, and not calling all these processings simple *computing*. But, why did not we introduce this distinction at the very beginning? Instead, we thoughtfully divided the field into finer subfields according to our growing understanding during the last decades.

- *Reduce it by abstraction!* This generally leads to modelling process we handle in detail in the next section.

Although the first step has been explained above scornfully thus in a provocative manner, it is of decisive importance: If we cannot carry out it properly, we cannot circumscribe the problem precisely. Consequently, we have to live with the fact, that complex systems cannot be immediately understood thoroughly, thus clearly described, fully evaluated, etc., otherwise they would have been simple, elementary, dumb systems, and not complex!

Modelling is a typical design activity supporting the systems engineer selecting the relevant features of the complex system to be developed. After the selection of these features, they must be specified in a wide sense, i.e. precisely protocolled to serve as input for the next design activity which can be carried out at the model, assuming we could create an appropriate model for the system we develop. This involves also means, e.g. methods and tools we need to enrich and complete the model adding heuristic knowledge we possess by experience that can be hardly represented entirely and algorithmically. These aspects will be handled in section 3, after we handle modelling aspects in section 2.

To assure that the system we built is in accordance with the user needs, we have to evaluate the implementation. Evaluation involves validation of user requirements, properly performed on the running (yet experimental) system in its expected environment under relevant, realistic conditions. Thus, evaluation can be a time consuming and costly process. Therefore, we have to accelerate and economize the evaluation process. This will be handled in sections 4 and 5. To illustrate the approach, an example for modelling and evaluation of a complex system will accompany the sections 2 and 3 to defend the provocative theses introduced in this paper. The author apologizes for not having prepared an example better related to the subject of this conference, i.e. public administration, but using an example introduced previously in [3].

As mentioned above, an experimental environment, revealing different aspects of evaluation and modelling will be utilized throughout the paper, introduced in the section 2.2.

As will be concluded in the section 4 and 5, the present approach is radically different than the conventional and widely accepted methods (see e.g. [11] in this volume): We focus on a formal model which will be operationalized incrementally. Moreover, the implemented system will be viewed as the result of this comprehensive modelling process, playing a central role in the evaluation of the system, e.g. to generate the test cases, test procedures, etc.

2. MODELLING

As indicated in the Introduction, the modelling step aims at reducing the complexity of development and handling of the system to be built. The reasons are evident: In the beginning, we focuse on *understanding* the subject (*psychological complexity*). Once we have understood what we are doing, or intending to do, we try to subsume the problems, e.g. categorizing them in order to make the system's complexity tractable (*computational complexity*).

The starting point to figure a model is usually given by behaviouristic approach: Here, we handle with the *external* (or synonymously, psychological) complexity. The behaviouristic model is often a shallow one; it is likely to lead us nowhere, e.g. to a dead-end concerning a useful solution, if this model cannot join (after appropriate refinements and supplements) the functional model which has to be very close to the system we wish to materialize - on condition that we built the *right* model, depending on our ability, skill and experience.

As mentioned repeatedly, modelling is a creative activity. It includes a design step concluded by the specification of the (possibly interim) results. These results are inputs of the next design step which can be viewed as a supplementary activity of the modelling process which virtually is to lead to the implementation - again, on condition that we work properly. In this sequel, modelling can be viewed as *operative prototyping* ([10]) to some extent. Modelling in this context is, however, more general than prototyping, as it additionally contains also elements of the conventional software development model (*waterfall model*), e.g. specification, evaluation, etc. and *transformation model* ([10]).

2.1. Algorithmic/Analytic Approach

Once the *right* model has been created, we can analyse it to validate user requirements, to understand its behaviour, operational structure, etc. Moreover, depending on the analysis results, we can extend the model.

The major group of the analysis methods are algorithmic, i.e. they can be deployed in accordance with the formalized features of the model. In other words, we can utilize these methods only when the problem field has been neatly and precisely formulated by formal specifications. The objective is then to manipulate the model if the analysis delivers unacceptable or undesired results concerning the user needs.

In the following sections, we demonstrate our approach by means of a practical example we borrow from [3] that will be modified and extended accordingly. We start the modelling process with a behaviouristic approach.

The described system is a component of a students' project at the Hochschule (Polytechnic) Bremerhaven, Department for Logistics, Transportation and Systems Analysis, running steadily since begin of eighties.

2.2. Case Study: Modelling a Multi-Storey Shelving System as a Practical Example

As a case study, we shall consider an automated commodity storage management system as utilized in many trading companies. In such a system, incoming commodities are stacked on pallets. The pallets, being automatically conveyed by the system, are then usually stored in a multi- storey shelving. The retrieval and the processing of the pallets is performed by the control system (usually materialized by a distributed system) which memorizes

- the contents of the pallet
- the location of the pallet in the multi-storey shelving.

As it can be seen easily, the example is not only non-trivial, but it also shows the typical features of complex systems we summarized in the beginning of the Introduction.

Whenever an order for supplying commodities occurs, the control system decides which pallet(s) shall be used to fulfil the request. If all the commodities on the pallet are required, the whole pallet is conveyed to the system's output point. Otherwise the pallet is conveyed to a commissioning robot for processing. The robot may remove certain commodities from the pallet and replace them with others (*commissioning* process).

To perform the mentioned tasks, the system comprises

- conveyor-belts to convey the pallets within the system
- turntables to change the direction of the pallets
- a commissioning robot
- a waiting circuit for pallets to be processed by the robot
- automatic forklift vehicles (flv) to access the multi-storey shelving
- transfer points where the pallets can be transferred to or from the forklift vehicles
- multi-storey shelving organized in parallel lanes to enable concurrent access to pallet locations.

The topological structure of such a system is shown in figure 1; the photograph in figure 2 displays the overall system and its components.

From the input point, a pallet may be forwarded to the shelving, to the waiting circuit, or directly to the output point, depending on the invoked operation to store, to commission, or to directly output this pallet. The pallets are distributed due to one of these alternatives by means of turntables in the central conveyor-belt, controlled by a real-time computer system. The conveyor-belt is in fact a series of belt-tables, each with a capacity of one pallet. Each table contains sensors which detect the presence of a pallet, read its identification code and control its conveyance either to another table, to the shelving or to the waiting circuit.

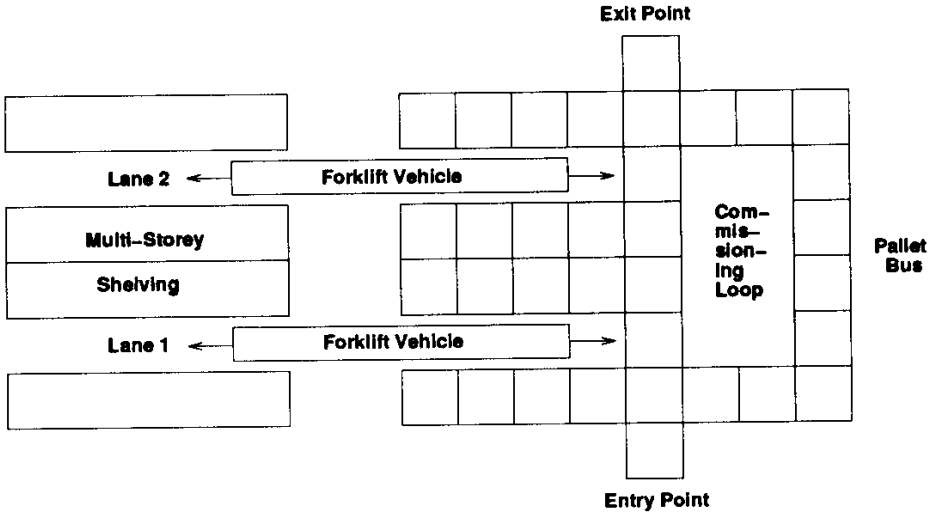

Figure 1. Structure of an automatized commodity storage management system (according to [5])

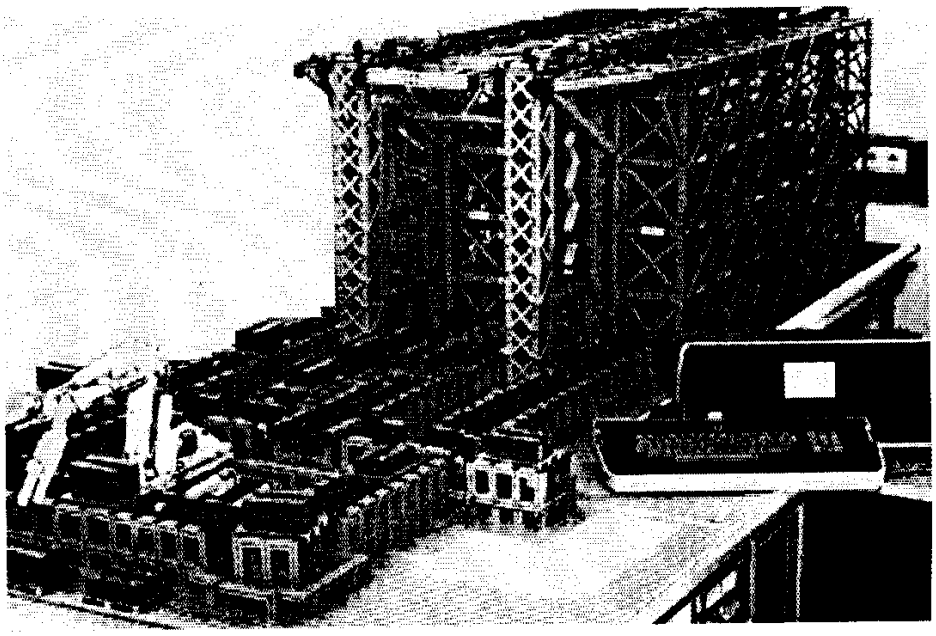

Figure 2. General view of the system to be modelled (The simulation model of an automated, commodity storage and multi-storey shelving system of the Hochschule Bremerhaven)

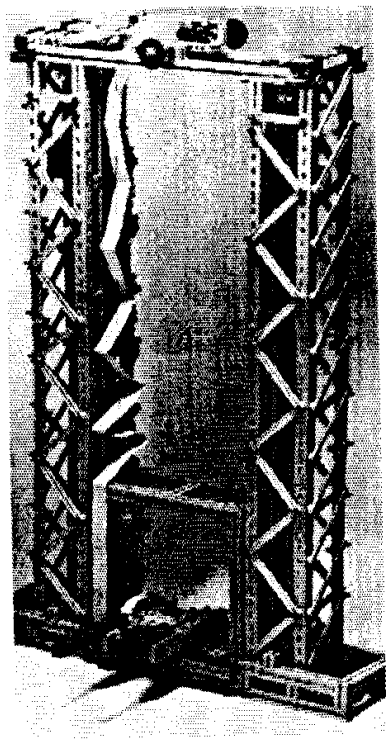

Figure 3. Forklift vehicle

2.3. Modelling the Overall Features by Predicate/Transition Nets

As an example, we will consider a precise, formal model of the processing of commodities in the multi-storey shelving system. Initially, a commodity is ordered by means of an order form $< o >$. The order form is sent to an external supplier, the copy remains with the ordering company. After delivery, the commodity is entered into the system on a pallet. For the pallet there is then issued an identification tag bearing the contents of the pallet in a code suitable for automatic character recognition, e.g. bar code reading. The pallet is then wrapped into a stretch or shrink foil film to avoid loss of its contents. Following that, a series of checks is performed whether these actions have been carried out properly. If one of those checks fails, the mentioned operations are retried, otherwise the pallet is automatically conveyed into the system (see figure 6).

In a hierarchical manner, we refine the component *Multistorey shelving* of the system represented in figure 6, yielding the nets depicted in figure 7. We arbitrarily quit the refinement procedure of this transition at this step as the achieved subnet is no more trivial but still lucid, i.e. it contains evident transitions.

As a next step, we further refine the component Pallet Processing and obtain a more detailed level as depicted in figure 6.

As a last step, we consider the further refinement of the transition Pallet Processing as represented in figure 9. Again, we arbitrarily quit the refinement procedure of this

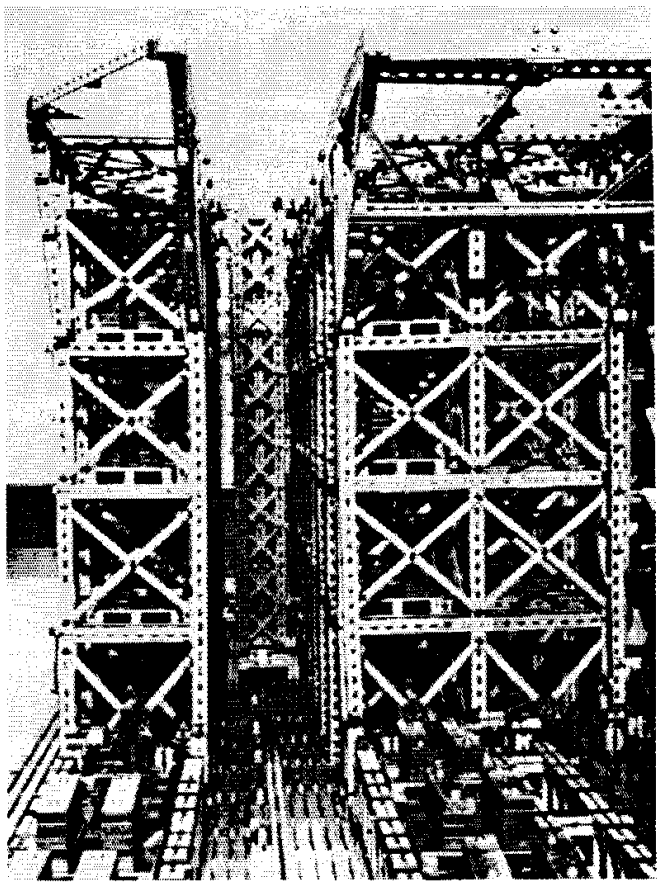

Figure 4. Multi-storey shelving lane

transition at this step as the achieved subnet is no more trivial but still lucid, i.e. it contains evident transitions. Moreover, the procedure described by figure 9 contains only transitions which will be operated sequentially, i.e. the entire process is a sequential one.

2.4. Refining the Model Interpretation

Generally, following alternative actions are possible once a pallet containing an ordered commodity has to be transported:

- The pallet is conveyed to and stored in one of the shelves.
- The pallet is conveyed to the commissioning robot.
- The pallet is conveyed directly to the output point.

The distribution of the pallets according to one of these alter- natives is performed by turntables in the central conveyor-belt system (predicate `PALLET BUS`).

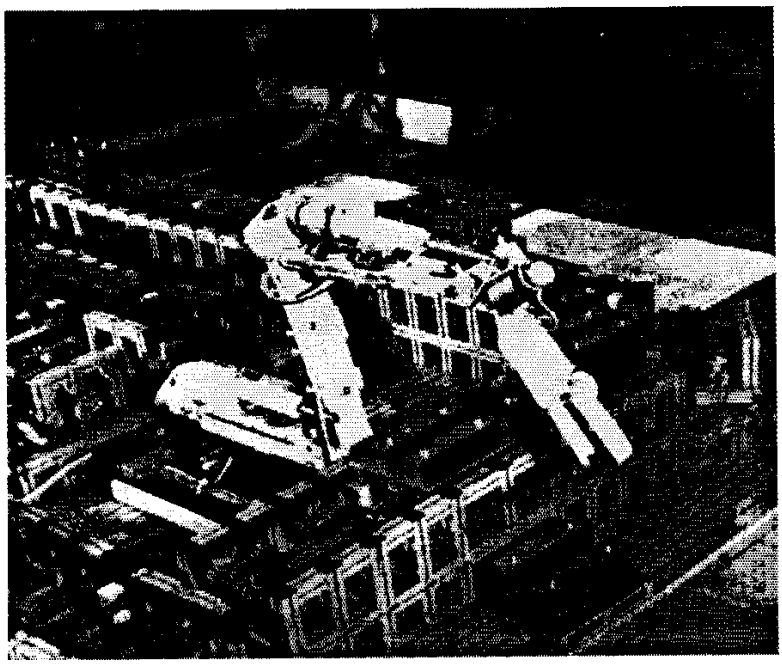

Figure 5. Commissioning robot

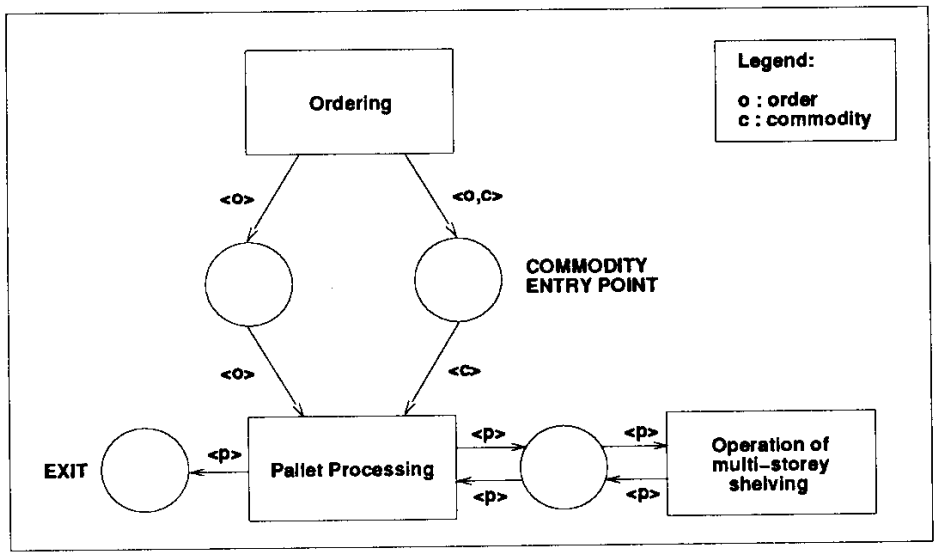

Figure 6. PrT net description of the system depicted in figure 2 (The topmost hierarchical level)

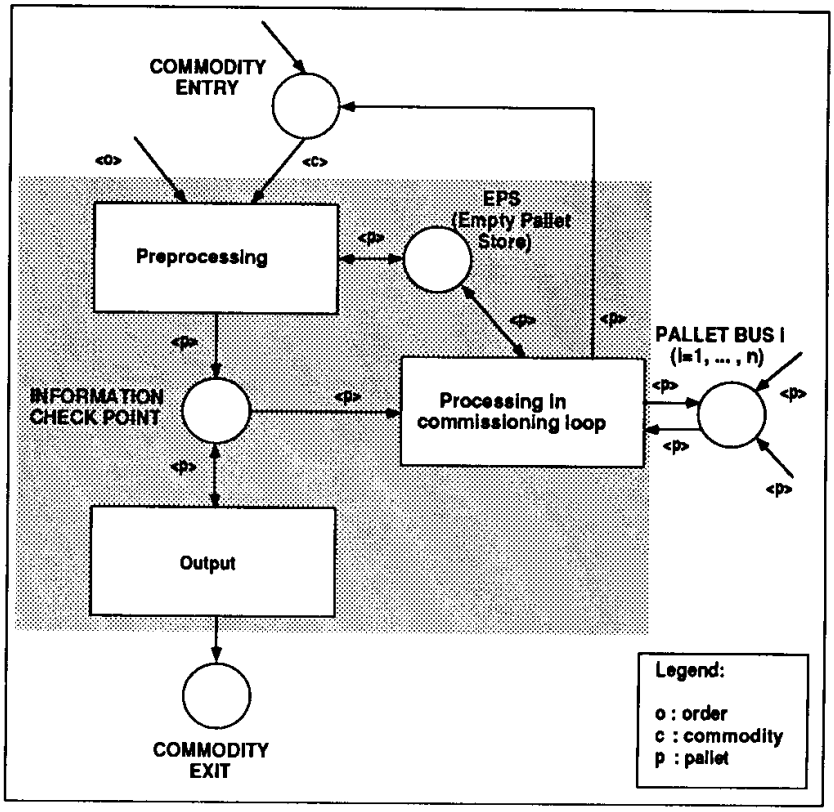

Figure 7. Refinement of the PrT net represented in figure 6 Refined transition: *Multistorey Shelving* (Example net component for invariant analysis in section 2.6)

Considering the case that the pallet, p, is to be stored in a shelf (transition INITSTORE), the following operations are performed:

First, the pallet is conveyed to a transfer point. If the forklift vehicle (flv) serving the requested shelving lane, l, is in position, it picks up the pallet. Simultaneously, it receives the location of the shelf from the control system, in which the pallet is to be stored. In our model, this is formally expressed by an address $< adr >$ consisting of the lane address, l, the rack address, r, and the number of the storey, s (It can be assumed that each shelf has a capacity of one and just one pallet). The forklift vehicle then moves to the designated location (predicate STORE ACCESS POINT). This is denoted by a formal sum:

$< p, adr > + < p > + < l >$

This formula represents that the pallet, p, is being transported by the forklift vehicle, l, with an order $< p, adr >$, and that p is to be placed at address, adr. At the STORE ACCESS POINT with the location l, r, s, the pallet is shelved. The fulfilled order is completed by a done message to the control system; this is represented by the tuple $< p, adr >$ flowing back to the predicate CONTROL. Subsequently, the forklift vehicle moves back to the initial transfer point (transition MOVE BACK).

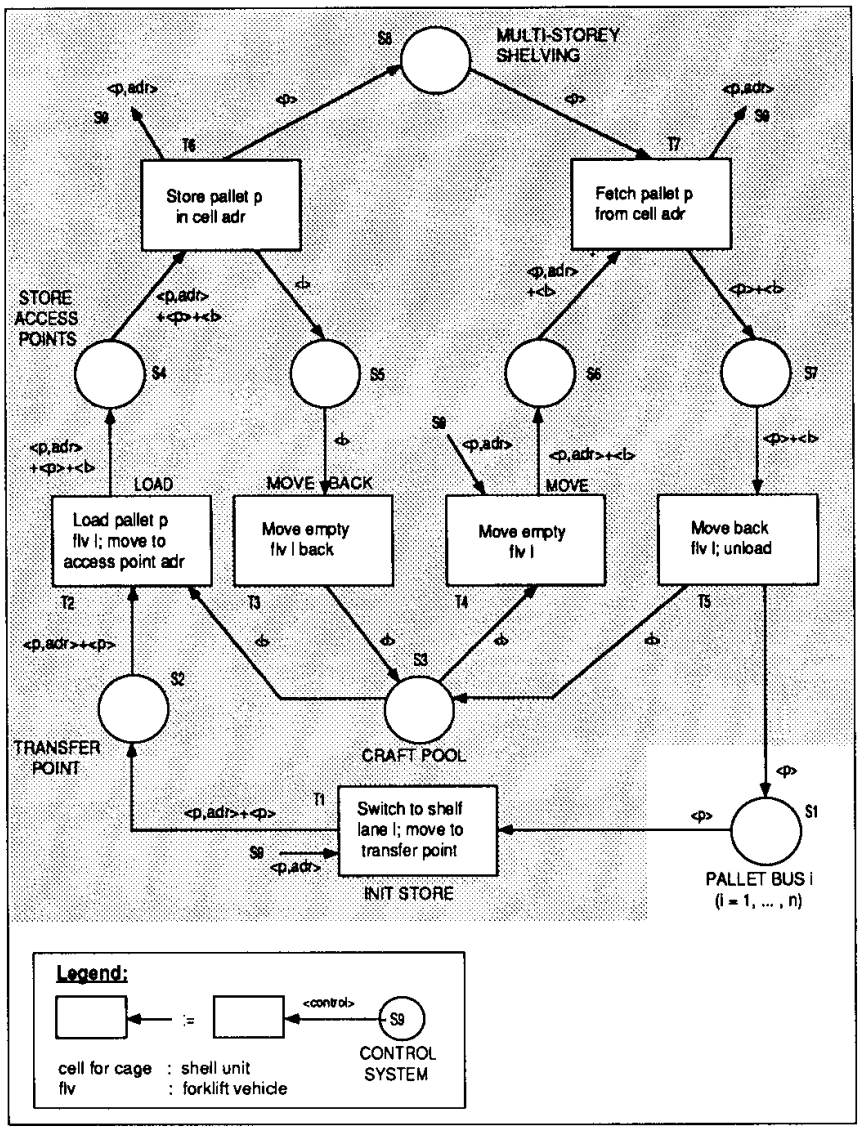

Figure 8. Further refinement of the PrT net represented in figure 6 refined transition: *Pallet Processing*

Correspondingly, if a pallet is to be retrieved from a shelf, the control system delivers the location $< l, r, s >$ of the pallet to the forklift vehicle which serves lane l (transition MOVE). After reaching the access point, the forklift lifts the designated pallet from its shelf and transports it back to the transfer point, from where it may be directed to another part of the system.

As the second main alternative for a pallet after entering the system, the pallet can be conveyed to the commissioning circuit. After reaching the commissioning robot, commodities may be removed from or placed upon the pallet. For and during this process, empty pallets may be required and generated respectively. It is therefore necessary to be

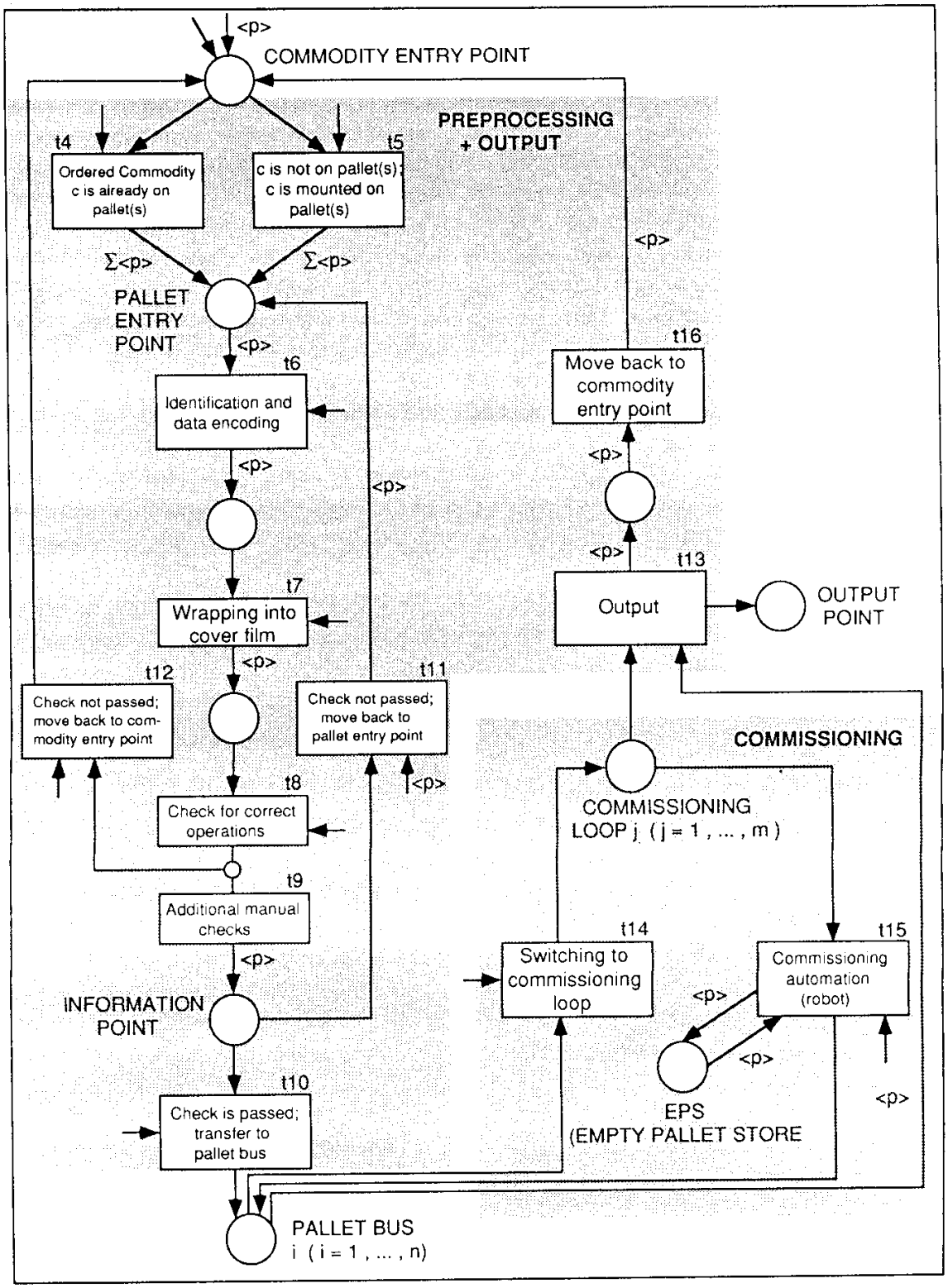

Figure 9. Refinement of the PrT net represented in figure 6 refined transition: Pallet Processing

able to convey empty pallets between the commissioning robot and the pallet rack (see figure 9).

It cannot be stressed too strongly that each predicate and each transition represents an entire multi-layer scheme of pallet states and state changes. So, an arbitrary number of independent pallets may concurrently flow between the predicates. The necessary synchronisation between pallets can very elegantly be expressed just by means of the capacities of the predicates. For a predicate name $i(i = 1, \ldots, n)$ representing a multi-layer of n simple predicates, for each of these components the capacity is just 1 (summing up over the index n then gives a total capacity n). The net specification automatically models the necessary constraint that in one location of the system (cell of the central belt, cell of the commissioning loop, shelf cell, etc.) at most one pallet can reside.

The operations of functional units in the system are controlled by means of operation codes distributed by the control computer into the units. This is modelled by an additional item $< control >$ flowing into every transition. As an abbreviation, in most cases this item is represented by an uninterpreted edge (see figures 7, 9).

A transition is enabled only when all input predicates carry the necessary items. For instance, transition LOAD of figure 7 needs the items $< p, l >$, $< l >$ and $< control >$. Before the transition starts, according to our approach moreover a check of the input items has to occur. Such a check can be performed by interpreting the items as an ordered string, e.g., $(p, l)(l)(control)$; this string can be modelled by regular expressions thereby systematically introducing check redundancy.

2.5. Analysis of the Modelling Net to Optimize the System Structure

In our model, some combination of items has been shown by a formal sum, e.g.

$< p, adr > + < p > + < l >$

as the output of transition LOAD (also named $T2$). Thereby, the range of the corresponding output predicates is given by the union of the disjoint ranges for the item variables

$< p, adr >, < p > and < l >$.

Distinguishing between these items can be performed without the restriction that these items have to occur properly in an order, e.g. as a *super-tuple*

$<< p, adr >, < p >, < l >>$.

	T1	T2	T3	T4	T5	T6	T7
S1	−⟨p⟩				⟨p⟩		
S2	⟨p,adr⟩ + ⟨p⟩	−⟨p,adr⟩ − ⟨p⟩					
S3		−⟨l⟩	⟨l⟩	−⟨l⟩	⟨l⟩		
S4		⟨p,adr⟩ + ⟨p⟩ + ⟨l⟩				−⟨p,adr⟩ − ⟨p⟩ − ⟨l⟩	
S5			−⟨l⟩			⟨l⟩	
S6				⟨p,adr⟩ + ⟨l⟩			−⟨p,adr⟩ − ⟨l⟩
S7					−⟨p⟩ − ⟨l⟩		⟨p⟩ + ⟨l⟩
S8						⟨p⟩	−⟨p⟩
S9	−⟨p,adr⟩			−⟨p,adr⟩		⟨p,adr⟩	⟨p,adr⟩

Figure 10. Incidence matrix of the subnet as depicted in figure 7 (consisting of the nodes $S1, \ldots, S9; T1, \ldots, T7$)

It should be noted that the items of the formal sum in every case are not completely independent of one another. As an example, for transition $T1$, there is the restriction that in order to enable the transition, the value of p in $< p, adr >$ coincides with that one in $< p >$; in the same way, for enabling $T2$ the value of $< l >$ has to coincide with that one of the component l of adr in $< p, adr >$ (see above). In most cases the items of the formal sums are tied together in a well-defined way; it is not possible that the item $< p, adr >$ is moved by a transition together with an item $< p' >$ which represents a pallet p' different from p.

Let us now demonstrate analytical net evaluation by the example of net invariances analysis. For reasons of simplicity, we shall confine to the subnet shown in figure 7. As abbreviations, the places and transitions of these subnets are additionally labelled $S1, \ldots, S9$ and $T1, \ldots, T7$, respectively. Figure 10 depicts the corresponding incidence matrix I which denotes adjacency between predicates and transitions: In this matrix, consider the case that a tuple $< a >$ flows from transition Tj to predicate Si, when Tj fires; this is denoted by setting the matrix element Iij to $< a >$; a tuple $< b >$ flowing from Sj to Ti is represented by

$Iji \doteq - < b >$

$$
\begin{aligned}
x1(-\langle \mathrm{p} \rangle) + x2(\langle \mathrm{p,adr} \rangle + \langle \mathrm{p} \rangle) + x9(-\langle \mathrm{p,adr} \rangle) &= 0 \\
x2(-\langle \mathrm{p,adr} \rangle - \langle \mathrm{p} \rangle) + x3(-\langle \mathrm{l} \rangle) + x4(\langle \mathrm{p,adr} \rangle + \langle \mathrm{p} \rangle + \langle \mathrm{l} \rangle) &= 0 \\
x3(\langle \mathrm{l} \rangle) + x5(-\langle \mathrm{l} \rangle) &= 0 \\
x3(-\langle \mathrm{l} \rangle) + x6(\langle \mathrm{p,adr} \rangle + \langle \mathrm{l} \rangle) + x9(-\langle \mathrm{p,adr} \rangle) &= 0 \\
x1(\langle \mathrm{p} \rangle) + x3(\langle \mathrm{l} \rangle) + x7(-\langle \mathrm{p} \rangle - \langle \mathrm{l} \rangle) &= 0 \\
x4(-\langle \mathrm{p,adr} \rangle - \langle \mathrm{p} \rangle - \langle \mathrm{l} \rangle) + x5(\langle \mathrm{l} \rangle + x8(\langle \mathrm{p} \rangle) + x9(\langle \mathrm{p,adr} \rangle) &= 0 \\
x6(-\langle \mathrm{p,adr} \rangle - \langle \mathrm{l} \rangle) + x7(\langle \mathrm{p} \rangle \langle \mathrm{l} \rangle) + x8(-\langle \mathrm{p} \rangle) + x9(\langle \mathrm{p,adr} \rangle) &= 0
\end{aligned}
$$

Figure 11. The equation system (corresponding to figure 10) for the derivation of the system invariants

The S-invariants of a net are solutions x of the equation system

$I'x = O$

where I' is the transposed of the incidence matrix I. The inner product of such a solution array x and a marking M of the net remains constant under a sequence of firings of transitions in the net [4, 7]:

$xM = xMO = const$ (M0: Initial Marking)

This feature provides a simple check whether a desired final state can be reached from an initial one by transitions of the net or not: The state can be reached if the constancy will not be violated, computed as the product of the array x and actual marking M.

2.6. Determination of System's Capacity through Invariant Analysis

The incidence matrix of figure 10 leads to the equation system of Figure 11.

A solution of the equation system represented in figure 11 is given by the array

$< x1, \ldots, x9 >=< 1,1,1, \ldots, 1 >$

As an example for the invariance of the product of marking and invariance array let us consider as an initial marking MO for the analyzed subnet the situation that two pallets p and p' are residing on $S1$, an item $< p, adr >$ (i.e. $< p, < l, t, s >>$, see above) is on $S9$ and an item $< l >$ is on predicate $S3$. It is evident that the product

$MOx =< p > + < p, adr > + < l >$

remains constant under the firings of the subnet. This inner product constitutes a simple check whether, starting from an initial situation, the subsequent states are correct; a failure of the system which could be seen either as an incorrect behaviour of a transition or the disappearance/appearance/changing of items on a predicate without firing a transition, would - at least with a high probability - also lead to a change in the inner product Mx and, thus, could be simply detected.

The presented example is a simple one with a very simple and elegant solution of the equation system. For more complex cases, special *projection techniques* [4, 7, 8] are proposed to simplify the equation systems derived from the incidence matrix.

3. INTRODUCING THE HEURISTICS

The methods introduced in the last sections, no matter how sophisticated they really are, or they will be felt by unpracticed, served solely one purpose: Modelling of exact features of the system. All steps are either of creative nature (creating the model, i.e. a design step), or specification of the design, i.e. protocolling the relevant features in order to invoke proper analysis techniques to verify critical characteristics. No matter how different these techniques are, they are common in their ability for being carried out algorithmically.

The techniques we deploy for the analysis are, as we could demonstrate in the last sections, powerful. According to the outcomes of the analysis, the next design step can lead to a manipulation of the model to consider the plurality of the requirements. Nevertheless, these techniques cannot solve the entire problem: We are very likely to possess further knowledge on the subject, e.g. experiences which are not precisely, but only vaguely to be specified and modelled. In other words, we need further means to represent and to analyze our heuristic knowledge.

The most severe requirement to be met when selecting techniques for the representation of heuristic knowledge is the compatibility of these techniques with the ones we have previously used within the algorithmic, first stage (see sections 2.1 and following). The latter must follow the former without gap, seamless.

The seamlessness is the reason we selected predicate/transition nets for the algorithmic/analytical stage of the modelling. These high-level Petri nets can be mapped directly to logic programs [12]. Logic programming with Prolog is a conventional way to materialize declarative and procedural knowledge [9]. Moreover, a logic program is never *final*, exactly the same way we never view our experience as perfect, i.e. complete/final. Accordingly, we can extend a Prolog program *in the fly*.

To demonstrate our approach, we first transfer one of the refined Petri nets of the last section (figure 7 and 10) to a Prolog program according to [3]. The predicates (s-elements) of the net, which figure the rows of the incidence matrix, figure also the predicates of the Prolog program. Positive elements of the matrix (figure 9) in a row are the conclusions, the negative ones correspondence to premises. The matrix elements themselves are the arguments of the predicates. Figures 7, 10, and 12 depict the net to be transformed, its incidence matrix, and the outcoming Prolog program, respectively.

The program we determined (figure 12) is very formal. Some *syntactical sugar* would make it more intelligible. We refine and extend some portions of this program, enrich it by comments, add some standard service programs, etc. obtaining a more realistic program (figure 13). This program contains our experiences in shape of constraints [2], e.g. to express the limitations when stacking commodities on pallets, or when generating plans to prepare customer specified pallets containing ordered commodities.

```
'S1'(P)       :- 'S7'(P), 'S7'(L).

'S2'(P,Adr)   :- 'S1'(P), 'S9'(P,Adr).          % 'S2'/2
'S2'(P)       :- 'S1'(P), 'S9'(P,Adr).          % 'S2'/1

'S3'(L)       :- 'S7'(P), 'S7'(L).
'S3'(L)       :- 'S5'(L).

'S4'(P,Adr)   :- 'S2'(P,Adr), 'S2'(P), 'S3'(L). % 'S4'/2
'S4'(P)       :- 'S2'(P,Adr), 'S2'(P), 'S3'(L). % 'S4'/1
'S4'(L)       :- 'S2'(P,Adr), 'S2'(P), 'S3'(L). % 'S4'/1

'S5'(L)       :- 'S4'(P,Adr), 'S4'(P), 'S4'(L).

'S6'(P,Adr)   :- 'S3'(L), 'S9'(P,Adr).          % 'S6'/2
'S6'(L)       :- 'S3'(L), 'S9'(P,Adr).          % 'S6'/1

'S7'(P)       :- 'S4'(L), 'S6'(P,Adr), 'S6'(L), 'S8'(P).
'S7'(L)       :- 'S4'(L), 'S6'(P,Adr), 'S6'(L), 'S8'(P).

'S8'(P)       :- 'S4'(P,Adr), 'S4'(P), 'S4'(L).

'S9'(P,Adr)   :- 'S4'(P,Adr), 'S4'(P), 'S4'(L).
'S9'(P,Adr)   :- 'S8'(P), 'S6'(P,Adr), 'S6'(L).
```

Figure 12. Prolog program corresponding to figure 7

```
customer_pallet(Customer,Order,Commissioning_plan) :-
   customer_pallet_gen(Order,[],Commissioning_plan).

% Generating a commissioning plan through "generate and test",
% utilizing accumulation technique.
customer_pallet_gen([],Partial_Commissioning_plan,Readymade_Commissioning_schedule)
   Partial_Commissioning_plan = Readymade_Commissioning_schedule.
customer_pallet_gen(Order,Partial_Commissioning_plan,Commissioning_schedule) :-
   generate_action(Order,Commodity_to_be_commissioned,
                   Partial_Commissioning_plan,Action),
   stack_constraint(Action),
   update_plan(Action,Partial_Commissioning_schedule,
                   Updated_Partial_Commissioning_plan),
   customer_pallet_gen(Commodity_to_be_commissioned,
                       Updated_Partial_Commissioning_plan,
                       Commissioning_plan).

% Generating an elementary action, according to available pallets
% and order
generate_action(Order,Commodity_to_be_commissioned,
                Partial_Commissioning_plan,Action) :-
   select_pallet(Partial_Commissioning_plan,Pallet),
   remove(Commodity,Order,Commodity_to_be_commissioned),
   commodity_on_assembly_line(Commodity,Conveyor),
   Action = move(Commodity,Conveyor,Pallet).

% Constraint for the pallet size, stack size
stack_constraint(Action) :-
   Action = move(Commodity,Conveyor,Pallet),
   stacksize(Height),
   height(Pallet,Pallet_height),
   Pallet_height < Height.

% Selecting a pallet to be stacked:
% 1) Pallet already existing, possibly not fully loaded.
select_pallet(Partial_Commissioning_plan,Pallet) :-
   member(Pallet,Partial_Commissioning_plan).

% 2) Generate new, empty pallet to be loaded
select_pallet(Partial_Commissioning_plan,Empty_pallet) :-
   Empty_pallet = [].

update_plan(Action,Partial_Commissioning_schedule,
                Updated_Partial_Commissioning_plan) :-
   Action = move(Commodity,Conveyor,Pallet),
```

```
   Pallet = [],
   Updated_Partial_Commissioning_plan = [[move(Commodity,Conveyor)]|
                                         Partial_Commissioning_plan].

update_plan(Action,Partial_Commissioning_schedule,
               Updated_Partial_Commissioning_plan) :-
   Action = move(Commodity,Conveyor,Pallet),
   select(Pallet,Partial_Commissioning_plan,[move(Commodity,Conveyor)|Pallet],
          Updated_Partial_Commissioning_plan).

remove(Commodity,[Commodity|Commodity_to_be_commissioned],
       Commodity_to_be_commissioned).
remove(Commodity,[Commodity1|Commodity_to_be_commissioned1],
       [Commodity1|Commodity_to_be_commissioned2]) :-
   remove(Commodity,Commodity_to_be_commissioned1,Commodity_to_be_commissioned2).

% Service procedures
height(Pallet,Height) :-
   length(Pallet,Height).

member(Head,[Head|_]).
member(Head,[_|Tail]) :-
   member(Head,Tail).

select(X,[X|Tail],Y,[Y|Tail]).
select(X,[Head|Xlist],Y,[Head|Ylist]) :-
   select(X,Xlist,Y,Ylist).
```

Figure 13. Prolog program containing constraints

4. INTERIM RESULTS

A chronological comparison of the different stages of the modelling process makes following thesis evident:

The implemented system, and consequently its implemented functions differ strongly from the former model. The original specification is obsolete.

This thesis articulates a natural phenomenon we are familiar with: The longer we study a system, or a process, the better we understand it. We often find out that our very starting concept was not appropriate, or simply wrong, leading the design process to a dead-end. A redesign, or restart is then the consequence.

Further, we summarize the experiment we carried out:

Predicate/Transition nets can be combined with logic programming for the deployment of both analytic/algorithmic methods and heuristic methods.

Finally, we repeat the fact that Prolog can be used for both, system specification (materializing declarative knowledge) and system implementation (materializing procedural and declarative knowledge).

5. SYSTEMS EVALUATION AND CONCLUDING THESES

Once the development has been concluded, the implemented system must be evaluated before it can be released, i.e. handed out to the user/customer. Evaluation of a system invokes following activities:

- *Validation of the user requirements* that the implemented system meets the user needs properly. Please note that we are not concerned here with verification aspects checking whether the tools and methods have been deployed properly when developing the system, e.g. utilizing the specification or programming languages and/or tools. Many testing methods have been developed to carry out the validation systematically, e.g. unit testing, integration testing, regression testing, etc. [6].

- *Reliability determination.* As the validation process is a time and resource consuming one, it can hardly be exhaustivly carried out. Nevertheless, releasing a system requires confidence on the trustworthyness of the system. A bundle of adjusted methods to determine the test effectiveness and coverage, thus to make a well-founded prediction about the system behaviour under specified conditions and time space, is given through reliability engineering using probability theory [1].

Evaluation (both validation and reliability determination) requires carefully prepared experiments virtually accelerating the evaluation process. These experiments represent scenarios condensing critical situations, thus reflecting long range experiences.

Test scenarios are also programs having procedural, but also declarative characters, reflecting our heuristic knowledge on the system under test and testing in general.

Following this argumentation, it is necessary to require that not only the system modelling and implementation, but also the system evaluation should be performed in a unified environment, as suggested here, in logic programming.

To run specific tests as dictated by the test scenario, one needs meaningful test cases, consisting of both test inputs to run the system and the expected outcomes of the execution. These expected (or intended) outputs will be compared with the actual outputs obtained during the experiments. It is evident that these test cases should be gained through the analysis of the implemented system.

We now can express our final theses:

- The development of the system can be viewed as a creative modelling process, i.e. the entire construction process can be carried out basing on the model.

- The implemented system can be seen as the ultimate system specification.

- System evaluation (validation and reliability determination) must be implementation-oriented. This implies that the test scenario, test procedures, and test cases must be gained through the analysis of the implemented system, and not of its specification, which is very likely obsolete when testing starts.

Further remarks, arguments, more methodologic aspects, and a uniform environment for an implementation-oriented evaluation can be found in [3] and [1]. These papers also

contain a formal representation of the subject and a rudimentary fault tolerance and test theory.

The author is aware of the fact that his above theses are contradictory to the "good" scholarship that is widely accepted and practised. He will appreciate a discussion critically reviewing the state of the art in software and systems engineering.

REFERENCES

1. Azem, A., Belli, F., Jack, O., *Implementation-Oriented Testing and Reliability Determination of Prolog Programs*, to appear in Proc. 5th International Conf. on Software Engineering and Knowledge Engineering, ACM Computing Press (1993)
2. Aiba, A. et al., *Constrained Logic Programming Language CAL*, Proc. International Conf. on 5th Generation Comp. Systems, pp. 263-276 (1988)
3. Belli, F., K.-E. Großpietsch, *Specification of Fault-Tolerant System Issues by Predicate/Transition Nets and Regular Expressions - Approach and Case Study*, IEEE Trans. on Software Engineering, 17(6), pp. 513-526 (1991)
4. Brauer, W. (ed.), *Petri Nets Central Models and Their Properties*, Berlin, Heidelberg, New York: Springer-Verlag (1987)
5. Buergin, K., *Design, Specification, and Implementation of a PC-Controlled Commodity Distribution Model: Pallet Conveying Subsystem* (in German), Diploma Thesis, Hochschule (Polytechnic) Bremerhaven (1985)
6. DeMillo, R.A., *Progress toward Automated Software Testing*, Proc. 13th International Conf. on Software Engineering, pp. 180-183 (1991)
7. Genrich, H.J., *Predicate/transition nets*, in Petri Nets Central Models and Their Properties, W. Brauer (ed.), Berlin, Heidelberg, New York: Springer-Verlag (1987), pp. 207-247
8. Jensen, K., *Colored petri nets are becoming a mature specification language*, in Proc. 10th Int. Conf. on Application and Theory of Petri Nets (Bonn), Berlin, Heidelberg, New York: Springer-Verlag (1989)
9. Lloyd, J.W., *Foundation of Logic Programming*, Springer-Verlag, Berlin etc. (1987)
10. McDermid, J. (ed.), *Software Engineers's Reference Book*, Chapter II/III, Butterworth/Heinemann, Oxford etc. (1991)
11. Olle, T.W., *Eight Viewpoints for Information Systems Methodologies*, FINAL, 3. Jg. 1/1993 (ISSN 0939-8821)
12. Peterka, G., Murata, T., *Proof Procedure and Answer Extraction in Petri Net Model of Logic Programs*, IEEE Trans. on Software Eng., vol. 15, no. 2 (1989), pp. 209-217
13. Zuse, H., *Software Complexity*, deGruyter, Berlin, New York (1991)