Modelling and Verification with Petri Nets‡

Comparing a State Oriented and Event Oriented Approach

Rainer Mackenthun
Fraunhofer-Institute
for Software and Systems Engineering
Berlin, Germany

Marc Voorhoeve
Eindhoven University
of Technology
The Netherlands

September 23, 1997

Abstract

A modelling and verification case study is presented and treated by two different approaches using Petri Nets. It is argued that these two approaches are in fact special cases of a common approach that takes into account all aspects of Petri nets.

1 Introduction

A key principle for Petri nets, presented in the system engineering book of the MATCH project [2] is the "The principle of duality". Nets consist of two disjoint sets of elements: state elements, representing passive objects of the modelled system and transition elements for modelling active objects. A *state* of the system being modelled is represented by a *marking* of the net model, an attribution of *tokens* to state elements. Similarly an *event* is represented by the *firing* of one or more transition elements.

A similar duality is present for Petri net modelling *methods*. One has *state oriented* and *event oriented* approaches. State oriented methods are biased towards the states (markings) of the modelled systems. Events (firings) are visible only insofar they modify states. Event oriented methods are biased towards the events of the system. States can be distinguished only by their event future. This paper compares a state-oriented and an event-oriented method by means of a case study and assesses the strengths and weaknesses of either method, indicating ways to combine them.

State oriented methods start with a specification of safety properties, restricting the possible markings and of liveness properties, restricting the set of possible marking sequences. The latter restriction excludes those sequences in which parts of the marking are stable even though progress is demanded for these parts. Typical notations for such specifications are the temporal logic approaches like [6] for the partial order semantics and [1] for the interleaving semantics. The MATCH-project focusses on interleaving semantics, so we use the UNITY approach of [1][1]. The first design steps implement the invariants using p-flows. Later on we implement the liveness properties by adding fair transitions. All development steps preserve the correctness of the model.

Event oriented methods describe the events that can take place during the system's existence. An event consists of the firing of a transition combined with the consumption and production of tokens. Transitions and places can be *labelled*; the firing of a labelled transition and/or the consumption and/or the production of tokens in labelled places are visible events. Events having the same labels are observed as equal. If all places and transitions involved in an event are unlabeled, the event is *silent* and cannot be observed. One might however discover the occurrence of a silent event by the observing that some hitherto possible future events have become impossible. By relabelling a net in such a way that different events in the old net obtain the same label, one

‡This work is supported by the European Union (EU) as part of the "Human Capital and Mobility (HCM)" project "Modelling and Analysis of Time Constrained and Hierarchical Systems (MATCH)" (CHRX-CT94-0452).

[1]We know the problems of the UNITY logic concerning the substitution axiom, but this does not affect our approach.

achieves *projection*. By partially unlabelling a net one achieves *abstraction*. Often the production and consumption of tokens is abstracted from in an event-oriented approach.

Typically, the modeller starts with a *specification* of the behaviour of the system w.r.t. an *environment*. This specification is often a simple net that omits events that are *internal* to the modelled system. The *implementation* shows these internal events and the engineer has to prove that the implementation satisfies the specification when internal events are abstracted from. Equivalence notions often used here are *weak bisimilarity* or the somewhat stronger *branching bisimilarity* ([3]), that incorporate a certain *fairness* assumption.

As case study we take a token ring mutual exclusion algorithm with a request mechanism. The algorithm is designed for n competing units. The informal description is as follows:

> If a competing unit has used the token and still possesses it and if it wants to reenter the critical section it sends a request onto the ring. If the competing unit gets a request it sends the used token to its neighbour.
>
> If a competing unit uses the token all incoming requests are delayed.
>
> If a competing unit has no token and it is not interested all incoming requests are relayed and the incoming token are also relayed. If the competing unit wants to enter the critical section it sends a request.
>
> If a competing unit has no token and it is interested the incoming token will be used and all incoming requests are relayed.

The described algorithm will now be developed in a state oriented and an event oriented way.

2 State Oriented Method

The development steps start with a collection of given temporal properties and a given environment. In this paper we will concentrate on the *invariants* and the *leads to* properties. A property "*a leads to b*" holds, if in every marking sequence a marking satisfying property a has a sequent marking satisfying property b. An invariant holds in every reachable marking. The environment has a fixed behaviour and is given as a Petri net. The example environment is given in figure 1. It is called client unit, since it describes the behaviour of a client interested in using the critical resources. The first c in the place names stands for *client*. A client unit is initially *not interested* (*cni*). When it gets *interested* (*cint*) it sends a *request* (*creq*). Getting a *permission* (*cperm*) it goes into the *critical section* (*ccs*). Leaving the critical section the client unit gets *not interested* (*cni*) while announcing being *ready* (*crdy*). All transitions will fire after the finite delay property except for transition *cgi* (*get interested*), since we do not want to demand from a client to become interested. In the following we will have n instances of competing units. Therefore the place names will be

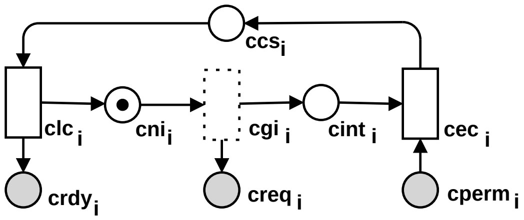

Figure 1: The client unit

indexed, e.g. ccs_i for the place ccs in the i-th competing unit. The temporal properties for the mutual exclusion algorithm are

$$\begin{array}{ll} \forall i \in \{1 \ldots n\}: & \mathbf{m}(creq_i) \geq 1 \text{ leads to } \mathbf{m}(cperm_i) \geq 1 \\ \forall i,j \in \{1 \ldots n\} \wedge (i \neq j): & \mathbf{m}(ccs_i) = 0 \vee \mathbf{m}(ccs_j) = 0 \text{ is invariant} \end{array}$$

where the first property denotes that a just requesting client i (at least one token on place $creq_i$) will always get a permission (at least one token on place $cperm_i$) and the second demands that the critical section will be used under mutual exclusion.

We now try to use p-flows to implement the invariants of the specification. Using p-flows we express linear invariants that are much more adapted to Petri nets than general invariants. By this step we may loose some possible solutions where specific invariants are not expressable using these special invariants. Nevertheless we get a wide area of solutions that are easy to implement.

There is no strict rule how to find these flows. Designing an algorithm is a creative process and needs some experience. Due to the lack of space we are not able to discuss that process. Here we explain the implemented properties.

In the below given invariants we have added some places that are used to describe certain properties of the system state. These place names have no additional initial c. The solution of the algorithm we have in mind is modular and lets the clients unchanged. The client i communicates with his local protocol via the channel $creq_i$, $crdy_i$ and $cperm_i$.

(1) $\forall i \in \{1, \ldots, n\} \,.\, \mathbf{m}(cni_i) + \mathbf{m}(cin_i) + \mathbf{m}(ccs_i) = 1$

(2) $\forall i \in \{1, \ldots, n\} \,.\, \mathbf{m}(ni_i) + \mathbf{m}(wt_i) + \mathbf{m}(cs_i) = 1$

(3) $\sum_{i \in \{1 \ldots n\}} (\mathbf{m}(cs_i) + \mathbf{m}(ut_i) + \mathbf{m}(tin_i)) = 1$

(4) $\forall i \in \{1, \ldots, n\} \,.\, \mathbf{m}(crdy_i) + \mathbf{m}(cperm_i) + \mathbf{m}(ccs_i) = \mathbf{m}(cs_i).$

(5) $\forall i \in \{1, \ldots, n\} \,.\, \mathbf{m}(cni_i) + \mathbf{m}(creq_i) = \mathbf{m}(crdy_i) + \mathbf{m}(ni_i).$

(6) $\sum_{i \in \{1 \ldots n\}} \mathbf{m}(wt_i) = \sum_{i \in \{1 \ldots n\}} (\mathbf{m}(rin_i) + \mathbf{m}(tin_i)).$

To model new properties and to be free in modelling changes of properties we can not use the places of the client since the client model should be unchanged. Therefore in the local protocol a state process is modelled that is strongly coupled with that of the client. Eqn. 1 describes the state processes of the clients. Eqn. 2 describes the state processes of the local protocols, where the abbreviations are as follows: ni_i for *not interested*, wt_i for *waiting for token* and cs_i for *in critical section*.

The coupling of the two state processes is given by eqns. 4 and 5: the protocol is only in its critical state cs if the client is in its critical state ccs (eqn. 4) and the number of not interested clients is given by the number of tokens on place ni of the protocol, if there are no "update messages" in $crdy$ or $creq$ (eqn. 5). Therefore place ni_i will contain a token when the instantanious knowledge of the local protocol is that its client is not interested.

Typically, in token algorithms the token is either used by one of the competitors (cs_i, *in critical section*), it is unused but still possessed by one of the clients (ut_i, *unused token*), or it is moving in one of the channels (tin_i, *token in*)(eqn. 3). Eqn. 4 together with eqn. 3 preserve the mutual exclusion property of the specification. The last equation restricts the activity in the net, as well as it guarantees activity unless all competitors are satisfied. There are always as many tokens in the channels (request or permission tokens) as competitors are waiting for the permission.

For the algorithm we will only use places contained in the linear invariants.

We have not yet restricted the direction of the transitions that we will add later on. This is sometimes necessary to enhance the modelling potentialities. Let us have a look on the client state process in figure 1. If we had to describe the process it could not only be done by eqn 1. There could have also been a transition from place ccs to place $cint$. To describe the orientation and the direction of the flow we add two kinds of restrictions. The first is the distribution restriction. This assigns transitions to a certain location and restricts the places the transition can consume from or produce for. The second kind of restriction excludes transitions between two given single places.

Distribution is given by the following function that is easy to understand. Each place is assigned to a pair of locations. Places with identical location elements belong to a single location. Places with two different location elements represent channels between the two locations. In our example we have $2n$ locations, n for the clients $((i, cl))$and n for the local protocols of the competitors$((i, pr))$.

$$\forall i \in \{1, \ldots, 4\} \,.\, \delta(p) = \begin{cases} ((i, cl), (i, cl)) & \textit{if } p \in \{cni_i, cint_i, ccs_i\} \\ ((i, cl), (i, pr)) & \textit{if } p \in \{creq_i, crdy_i\} \\ ((i, pr), (i, cl)) & \textit{if } p \in \{cperm_i\} \\ ((i, pr), (i, pr)) & \textit{if } p \in \{cs_i, ni_i, wt_i, ut_i\} \\ ((i, pr), (i \oplus 1, pr)) & \textit{if } p \in \{tin_{i \oplus 1}\} \end{cases}$$

$i \oplus 1$ denotes the next competitor on the ring. Transitions can only connect two places if the second location element for the starting place is the first location element for the ending place.

As an example for the second kind of restriction we forbid the moving of tokens from place cs_i directly to place $tin_{i \oplus 1}$ and from place tin_i to place ut_i for design reasons.

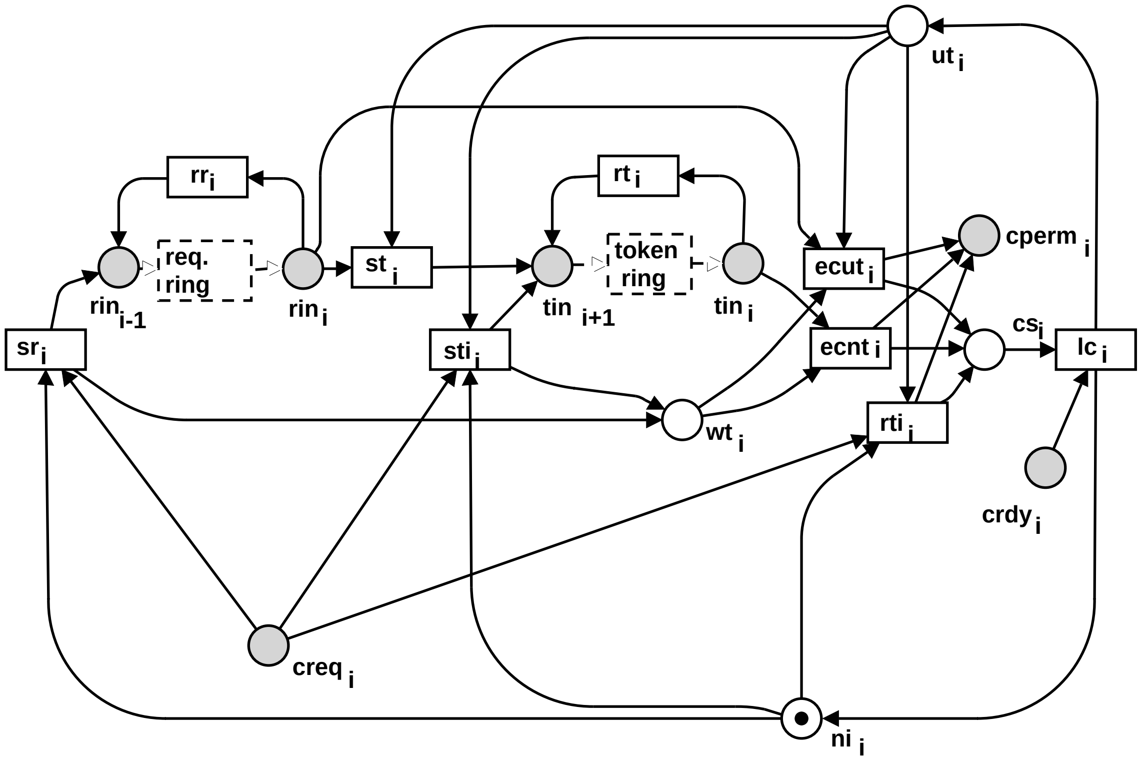

Figure 2: Request token ring - first version

Now we add all transitions (see restrictions below) that preserve the invariants and properties mentioned above, to fulfill the liveness properties. We assume transitions to fire strong fair (the strong fair transition cannot be infinitely often enabled without firing). This assumption is very strong but we use it in this step to postpone a detailed discussion of suitable fairness properties. They will be discussed at the end of this section. Figure 2 shows the protocol of one competitor. The grey places will be put together by place fusion with the corresponding (equally named) grey places in other parts of the model e.g. local protocols of other competing units and clients. In the figures we write $i-1$ instead of $i \ominus 1$ and $i+1$ instead of $i \oplus 1$ due to limited modelling facilities of the used tool. The events can now be identified. Transition *rr relays* a *request*, Transition *rt relays* a permission *token*. *st* allows a request and *sends* the used *token* to its neighbour. *rti reuses* the used *token immediately*, if the client requests again for the token. *sr sends* a *request* for the token if the client requests again for the token. *sti sends* the *token immediately* without sending a request if the client requests again for the token. *ecut enters* the *critical* section with an already *used token* when a request arrives and the competitor is waiting for the token. *ecnt enters* the *critical* section with a *new token* when it arrives and the competitor is waiting for the token. The dashed boxes named *request ring* and *token ring* represent the fact, that the request and permission token will be transported by the other units from the input places of the neighbour back to the own input place. They are only comments and have no formal semantics.

We have to add all possible transitions, except for those excluded by reduction rules. To explain the reduction see figure 2. We could have added a transition that consumes tokens from tin_i and rin_i and produces tokens for $tin_{i\oplus 1}$ and $rin_{i\ominus 1}$. This would preserve all properties. This transition is omitted since the same incidence can be achieved by concurrent firing of rr and rt. We could also add a transition consuming tokens from ni, tin, and $creq$ and producing tokens for cs, $cperm$, and $rout$. The transition is omitted since the same incidence can be achieved by firing first sr and then $ecnt$. So we can omit a transition if its incidence vector is a linear combination of other, already included transitions.

For further development we use a proof graph similar to the one used in [5]. At first, it can be used to prove *leads to* properties of our system.

To understand the idea of the proof graphs see figure 3. This is a first preliminary version of the proof graph for a system with two competing units. The nodes of the proof graph represents subsets of the reachable markings, e.g. $creq_1$ represents all markings m where $\mathbf{m}(creq_1) \geq 1$. These

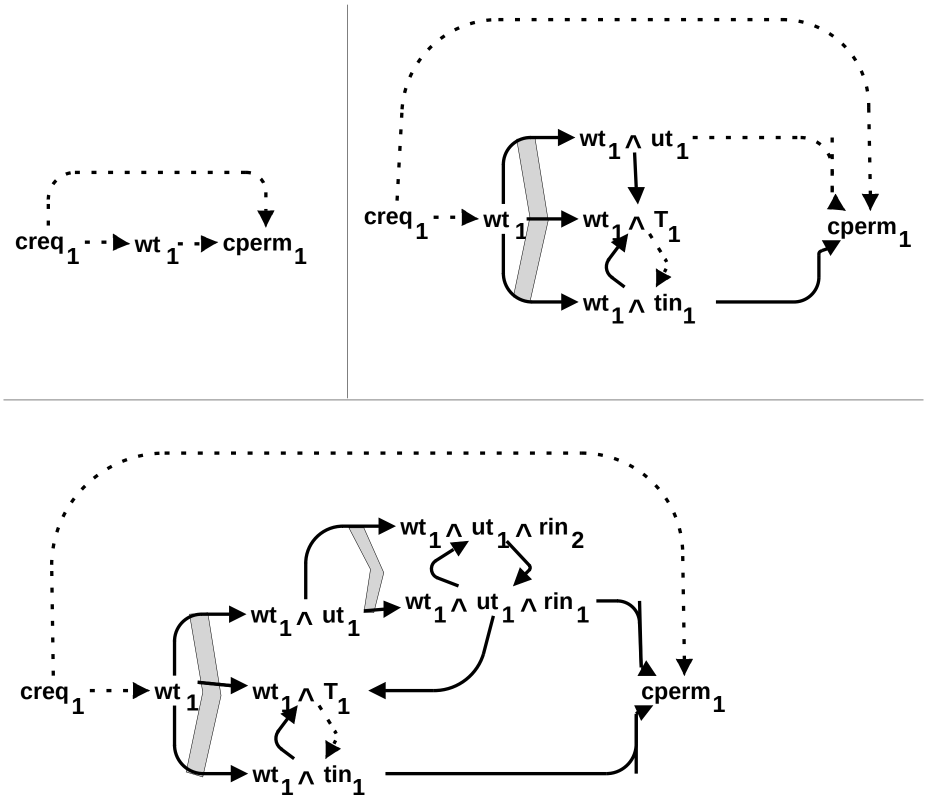

Figure 3: Proof Graph - Part I

markings are called $creq_1$-markings. Since all places are 1-safe except for rin_1 and rin_2 we can interpret for instance $creq_1$ as $\mathbf{m}(creq_1) = 1$.

The property we want to prove is "$\mathbf{m}(creq_1) \geq 1$ leads to $\mathbf{m}(cperm_1) \geq 1$". A proof graph is developed in two steps. At first we want to know which markings can be reached if we start firing transitions in a $creq_1$-marking and terminate the firing sequence if a *cperm*-marking is reached. Secondly we examine if the transitions between markings are guaranteed.

The first problem we have to solve is the problem of the granularity of the proof graph. How can we find suitable subsets of markings that allow us to proof the property and at the same time to keep the complexity as low as possible? Therefore we start the development by the partition that is given by the invariant $\mathbf{m}(cni_1) + \mathbf{m}(creq_1) + \mathbf{m}(wt_1) + \mathbf{m}(cperm_1) + \mathbf{m}(ccs_1) = 1$, a linear combination of eqn. 2, 4, and 5. By that invariant we know that a marking is either a cni_1-, a $creq_1$-, a wt_1-, a $cperm_1$-, or a ccs_1-marking. This invariant resp. partition is chosen since it contains $creq_1$ as well as $cperm_1$.

The upper left graph in figure 3 presents the result. From a $creq_1$-marking we only reach wt_1-markings and $cperm_1$-markings directly, from a wt_1-marking we only reach $cperm_1$-markings directly. The dotted arc means that it might be possible to reach the respective set of markings directly, but we are not sure that it will be reached since all transitions have additional enabling conditions. For instance transition sr_1 is only enabled in a $creq_1$-marking if ni_1 contains a token, too. So even the assumed strong fairness does not yet guarantee progress, if we focus on this "high level" proof graph.

Reachability is not the property we want to prove. We want to be sure that some transitions will occur. This can only be proved by refining the sets of markings in the proof graph. In the right upper graph we have partitioned the wt_1-marking into three subsets. The partition is complete due to the fact that a competitor cannot simultaniously wait for a token and use it (eqn. 2 and 1) and due to eqn. 3. The token is either unused, in a channel or used by one of the competitors. The later is stated by T_1 that is an abbreviation for $ut_2 \vee tin_2 \vee cs_2$. This complete partitioning is represented by the arcs connected by the grey ribbon. In this graph we have solid arcs that ensure transitions between the corresponding nodes. In all $(wt_1 \wedge tin_1)$-markings rt_1 and $ecnt_1$ are enabled. Due to the fairness assumption one of these transition will fire in a $(wt_1 \wedge tin_1)$-marking and the resulting marking will be a $(wt_1 \wedge T_1)$- or a $cperm_1$-marking. Another refinement is presented in the lower graph in figure 3.

The goal is to find a refinement, where the node $cperm_1$ can be reached by solid arcs only from all nodes that are reached when firing the above mentioned firing sequences from $creq_1$-markings. If we can construct such proof graphs for all demanded *leads to* properties we have finished our proof.

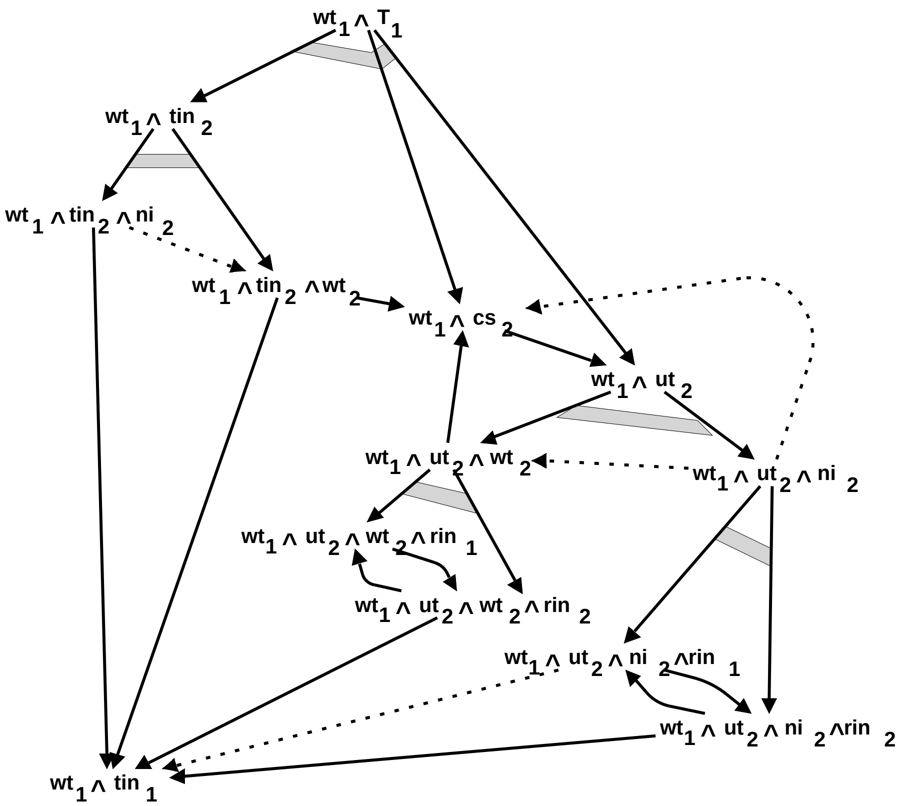

Figure 4: Proof Graph - Part II

These graphs can be rather complex. In figure 4 we give the part of the final proof graph that replaces the graph transition from $wt_1 \wedge T_1$ to $wt_1 \wedge tin_1$ in the lower graph in figure 3. A detailed description is omitted. We also omit the refinement for the transition from $creq_1$ to wt_1, but the result of the full proof graph is that the net in figure 2 fulfills the *leads to* property of the specification.

If the demanded property cannot be proved, we must go back to the safety development steps and change the solution ideas and their consequences. If the property can be proved we use the proof graph for another task. There might be graph transitions that are optional since the *leads to* property is also preserved without them. In this case it is possible to remove net transitions if the corresponding graph transitions are optional. By removing different sets of transitions we get different (classes of) algorithms. By removing the optional transitions *sti*, *rti* and *ecut* we get the net in figure 5 that represents a preliminary version of the algorithm we looked for. The *leads to* property is still valid.

The last step concerns the fairness of the transitions. Until now we have assumed strong fairness for protocol transitions. Strong fairness is not easy to implement and in many cases unnecessarily strong. We only need strong fairness to get out of a cycle in the proof graph. But even in this cases the strong fairness can be replaced by the productive firing rule (a productive transition cannot be permanently enabled without firing) if we add some additional side conditions that result in a removing of arcs in the proof graphs and break the cycle.

We explain this aspect using figure 5. If the places rin_i and ut_i contain a token, transition rr_i and st_i are in conflict. There is a cycle in the proof graph representing the moving of requests tokens along the ring. So the transition st_i must fire strong fair, since otherwise the token would be able to move around the ring forever. By adding a side condition to transition rr_i the firing of st_i is guaranteed and we do not need the strong fairness any longer (see figure 6). The side condition can be extracted from the proof graph.

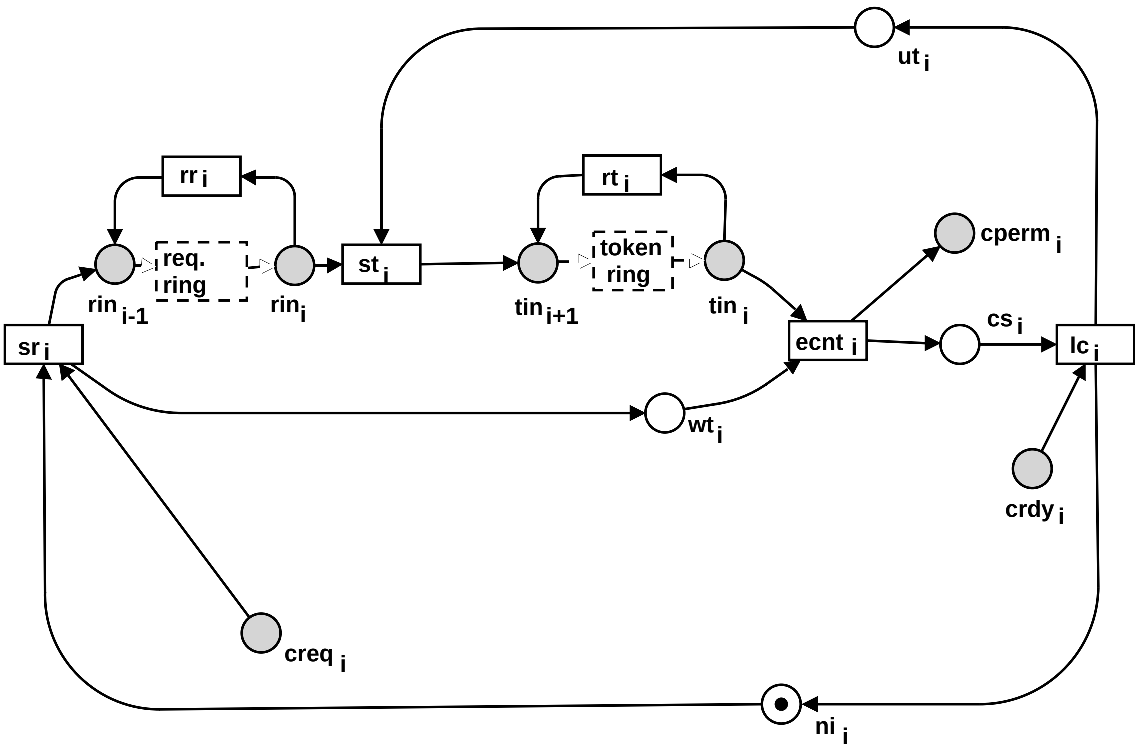

Figure 5: Request token ring - intermediate version

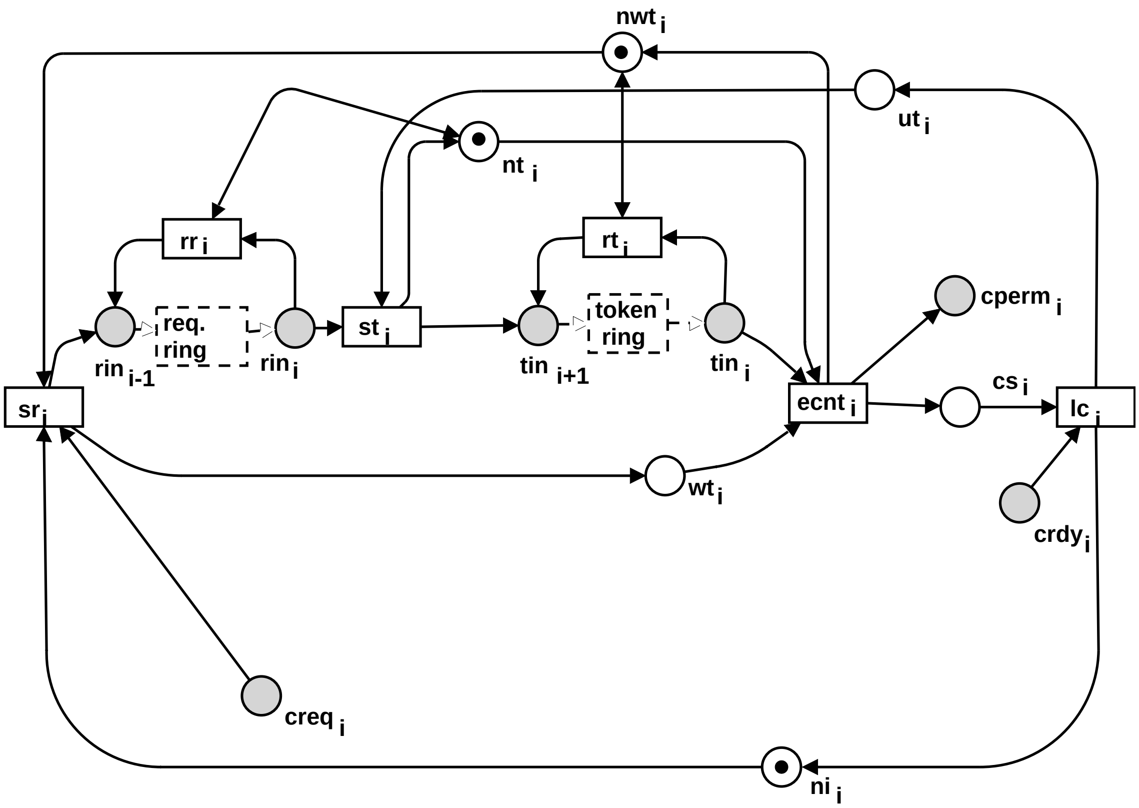

Figure 6: Request token ring - final version

The idea is illustrated by transition rr_2. That transition is responsible for the graph transition between the nodes $wt_1 \wedge ut_1 \wedge rin_2$ and $wt_1 \wedge ut_1 \wedge rin_1$ in figure 3 as well as for the graph transition between the nodes $wt_1 \wedge ut_2 \wedge wt_2 \wedge rin_2$ and $wt_1 \wedge ut_1 \wedge wt_2 \wedge rin_1$ and between the nodes $wt_1 \wedge ut_2 \wedge ni_2 \wedge rin_2$ and $wt_1 \wedge ut_1 \wedge ni_2 \wedge rin_1$ in figure 4. All these three are optional. But net transition rr_2 cannot be removed since it is used for mandatory graph transition in the proof graph for property "$\mathbf{m}(creq_2) \geq 1$ leads to $\mathbf{m}(cperm_2) \geq 1$". So we have to find such a side condition that the enabling condition includes the preconditions of mandatory graph transitions and excludes preconditions of optional graph transitions, that produce cycles. In our examle we restrict transitions rr_2 to the cases, where no token is possessed by competitor 2. By this the mandatory action that prevents a blockage of a request at a never interested competitor 2 is preserved while the optional possibility of sending the token again on the ring even though it could be used for an activation of the token (transition st_2) etc. is omitted.

Figure 6 is the final solution of the development process.

3 Event Oriented Method

The starting point of event oriented Petri net modelling is identifying the events that matter to an environment of the system to be modelled. The natural environment of our case study system is a user of a node in the mutual exclusion token ring. The events that matter are the firing of the transitions `crq`, `cp` and `crd` that respectively request admission to the critical section, give permission of entering the critical section and announce exiting the critical section. The order in which these actions appear is always the same and in the initial state the user starts with event `crq`.

The specification is thus given by the net in Figure 7. Instead of this net specification, one can use a proposition in a modal language like HML (cf. [4]).

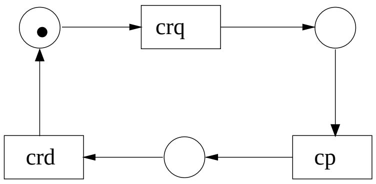

Figure 7: Request token ring - first specification

From this specification one infers that a user may expect that his requests for entering the critical section will be honoured at some time in the future and that he may not issue another request before having exited. However, the exclusive access is not guaranteed. To this end, we present a specification with two arbitrary users n and m in Figure 8.

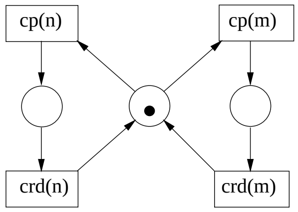

Figure 8: Second specification

This specification exhibits the exclusive access, due to the single token in the central place. Note that these specifications are common to every mutual exclusion protocol.

We can arrive at a common abstraction of the two specifications by abstracting from the request in Figure 7 and from, say, user n in Figure 8, after relabeling the nets. The resulting nets in Figure 9 are weakly (and branching) bisimilar, so the two specifications are consistent.

Next, we implement the request ring. First an arbitrary node is modelled, then a number of identical nodes is connected. The node model consists of the three-event loop of the specification with some extensions. The `cperm` action needs the token in order to fire. When `creq` fires, a request for the token is issued. If a request from another node comes in then the token is released (event

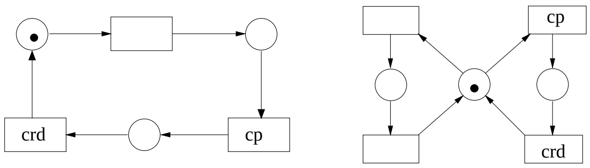

Figure 9: Common abstraction of specifications

st) if the node holds the token, otherwise the request is relayed (event rr). If the token comes along and the node is not interested, the token is relayed (event rt). In order to prevent a node from monopolising the token, the token has to be released before it can be obtained again. This behaviour of a node is modeled in Figure 10. A node has two possible initial states: in the one depicted neither the token is present nor needed. In the other state the token is present but not needed. This is state reached after crd has fired.

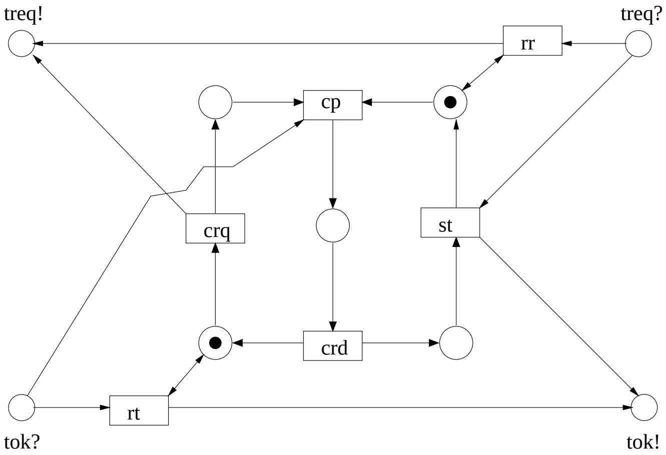

Figure 10: Implementation of a node

Note the similarities and differences with Figure 6. Instead of the places creq, cperm and crd for the (asynchronous) communication with the client, there are events with the same name that the client can synchronise with. Also the place $\overline{\texttt{wt}}$ from that figure is missing. Instead, the rt transition needs the place ni to be marked. Since the ring contains but one token, this difference is irrelevant to the behaviour of the node.

Now the nodes are combined in a ring as indicated in Figure 11. The tout and rin places of a node are fused with respectively the tin and rout places of its right-hand neighbour. Initially one node, say node(0) is holding the token, the others are not.

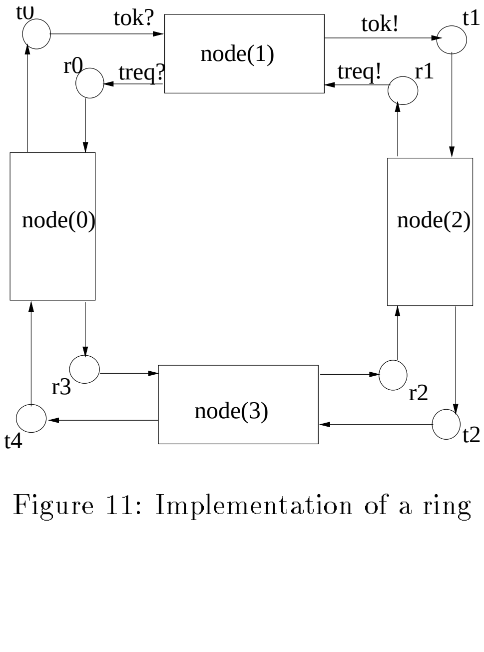

Figure 11: Implementation of a ring

We now want to verify that the implementation satisfies the specifications. To this end we verify that the ring consisting of k nodes is weakly bisimilar to Figure 8, respectively Figure 7 after the proper abstracting and renaming. For small k this can be done automatically. This will boost confidence in the implementation.

For larger k, one has to resort to ad-hoc methods and actually *construct* weak bisimulations. For the present case, the bisimulations are straightforward; the only problem is to prove that they actually *are* weak bisimulations. The essential step in this proof is to show that after `creq` has fired (and thus a `rout` request is issued), it is guaranteed that `cperm` will fire (so the `tin` token will be produced). This is a kind of *liveness* result that can be proved by induction on the number k of nodes in the ring.

4 Evaluation and conclusion

Both approaches sketched are able to support the modelling process by departing from specifications and building implementations that satisfy them. The state-oriented approach uses assertional techniques and the event-oriented one uses an equivalence relation (weak bisimilarity). Assertions are helpful in deriving an implementation from the specification, whereas equivalence is oriented towards verification of a given implementation.

Event-oriented assertional methods do exist, based on e.g. HML [4]. On the other hand, one can derive equivalence relations based on states rather than events. The main difference between state and event oriented approaches lies in the fact that one abstracts from events and the other from states as much as possible. Events are reduced to a mere reachability relation between states in the one and states are reduced to a future behaviour in the other.

Concentrating on one aspect of systems has the advantage of simpler notions and constructs. However it also restricts the freedom of the modeller, who would like to reason about both states and events.

The state oriented approach suffers from the difficulties that the engineer has to overcome in finding an adequate set of places and constructing p-flow equations from the informal description. It would be an improvement to start by identifying events of the system. This would help the engineer to understand the algorithm. After that, he can use the systematic state oriented approach to construct a correct algorithm and even to compare it to other possible solutions that the systematic approach will produce.

Likewise, the event oriented approach would benefit from a possibility to identify states. The only way to characterise a state of a system is by means of its future behaviour. So one is compelled to resort to temporal statements like "the node is in a state in which an incoming token request will be relayed". Such properties of the state are global and cannot be localised in any natural way. It is very natural to reason about tangible and local aspects of its state, like "the node is not in possession of the token", which can be easily modelled by a place being marked.

The simple system in our example does not suffer too much from these disadvantages. However, as systems become more complex, a pure state or event-oriented approach will not be enough.

Since Petri Nets possess both state and event components, a true Petri Net modelling approach should be a combined one. This is still a research subject, but the following ideas come to mind.

- A state/event logic to reason about a system's state as well as about the events in its future, generalising both one-sided logics like UNITY and HML.
- A bisimulation equivalence that takes visible aspects of a system's state into account.
- The possibility to project/abstract from both state and event components.

The two approaches in this paper then would become special cases of the combined approach, where the first abstracts from events and the second from states.

References

[1] K.M. Chandy and J. Misra. Parallel Program Design - A Foundation. *Addison-Wesley*, 1988.

[2] C. Girault and R. Valk (ed.). System Engineering - A Petri Net Based Approach to Modelling, Verification and Implementation. *(to appear in 1999).*

[3] R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.

[4] M. Hennesy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[5] R. Walter. Petrinetzmodelle verteilter Algorithmen. Beweistechnik und Intuition. *Band 2 aus Edition Versal, Bertz Verlag, Dissertation, Humboldt-Universität zu Berlin* 1995.

[6] R. Walter, H. Völzer, T. Vesper, W. Reisig, E. Kindler, J. Freiheit und J. Desel. Memorandum: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. *Informatik-Berichte der Humboldt-Universität zu Berlin* No. 67, August 1996.