Process Transition Networks: A Formal Graphical Knowledge Representation Tool

Jacek Malec*
Department of Computer and Information Science
Linköping University
S–581 83 Linköping, Sweden
jam@ida.liu.se

Abstract

A formal definition of Process Transition Networks (PTNs), recently introduced by Sandewall [San90], is presented. PTNs may be used as a graphical language to describe complex, qualitative processes; processes, which are difficult to express quantitatively even with the help of causal dependencies. The formal definition of the language is based on graph theory, and has strong similarities with Petri net theory. PTNs are first introduced informally, followed by syntactic and semantic specifications. The relation between PTNs and Petri nets is considered, and it is shown that although PTNs have no more expressivity than Petri nets, their flexibility as a representation language for complex processes justifies their existence.

1 Introduction

In this paper I present Process Transition Networks (PTNs), recently introduced by Sandewall [San90]. They are intended to serve two purposes:

- make the knowledge acquisition process easier (much less effort is needed to construct PTN diagrams describing particular traffic situations than with other representation tools, e.g. logic formulae, or Petri nets);
- serve as a tool for knowledge explanation to non-experts.

The number of existing knowledge representation tools, including several graphical languages, raises the question as to whether yet another graphical knowledge representation tool is necessary. This paper can be seen as an attempt to (indirectly, but positively) answer this question. First, the language is defined formally (both its syntax and semantics), which makes it possible to compare it with other similar tools, and (in the best case) to prove its useful properties. Second, PTNs appear to be well suited for expressing common sense conceptualizations of complex environments, especially in the case where causal dependencies between described entities are involved.

Although we claim that PTNs might be useful in several different application areas, we currently use them only for expressing knowledge about typical traffic situations. They serve as a tool for transferring knowledge from "domain experts" (which, in case of traffic scenarios, all the drivers feel to be ones) to a driver support system, currently being developed in our lab. As we use several knowledge representation tools within the system, the possibility of moving freely between representations becomes an additional advantage. In this context, clear definition of syntax allows us to treat PTNs as the specification language for system development, and its

*This research was supported by the IT4 research program.

formal semantics provides intuitions about the relation between this language and logics used elsewhere in the system.

PTNs are closely related to several well-known formalisms which exploit the advantage of graphical representation. One of the best known formalisms are Petri nets [Pet81]. PTNs are a proper subset of Petri nets, but they have been designed in such a way that expressing causal (and thus temporal) dependencies requires only one symbol, which is impossible to obtain using Petri nets. Moreover, the properties of Petri nets have been well studied, so we hope to get more insight into PTNs by establishing a formal correspondence between the semantics of both languages.

Another well designed graphical formalism has been developed by Harel [Har87]. His *statecharts* seem to have motivations similar to PTNs. Harel also provides a formal (although only operational) semantics for statecharts. However, his approach seems to have one major drawback as compared with ours (at least within our application area). Statecharts are well suited for describing entities fitting the automata theory paradigm. This implies that one has to distinguish an input alphabet (set of events), states of described entity, and a transition function mapping current state and some input value to the new state (and possibly to the output). In our case we want to mix dependencies of the form $input \rightarrow state$ with dependencies $state \rightarrow input$, where *state* correspond to some state of a (possibly intelligent) agent, and *input* is a description of current state of the environment. In the automata theory paradigm, the set of input letters, and the set of output letters are totally unrelated. Of course, very close links tie both formalisms, so it might be interesting to look at their relationship in more systematic way.

The paper consists of five sections. Section 2 describes PTNs informally, to convey their intended meaning, but without introducing unnecessary details (the interested reader is asked to consult [San90, Mal91b][1]). In section 3 a formal definition of a PTN as a graph is given. Section 4 contains an informal description of the PTN semantics (formal definition is presented in [Mal92]). Finally, in section 5, we show how a PTN can be reformulated as a Petri net.

2 Process Transition Networks

Let's begin with a bit of ontology. We assume that the world consists of objects (some of which we may call *agents*), and that the state of the world may be described by using a set of object properties. Property values may (and usually do) change with time. We introduce in the PTN language a box representing a *process*, or formally, a set of time tokens during which some property *continuously* changes its value. We are also interested in those moments in which discontinuities occur, because such discontinuities convey important information about current situations. Being more precise, discontinuities inform us about important situation changes. We will call such discontinuities *events*. Events are represented in the PTN language by hexagonal boxes. As discontinuities (by definition) may only appear in between periods of continuous change of some property, events may only appear between two processes. A *transition* (from process to event to another process) is denoted by two thick arrows pointing from the first process to the event, and from the event to another process, respectively. Causal dependencies between discontinuities are represented by *enabling arrows*, shown in figures with thin lines, and pointing from either an event alone, or an event and its subsequent process, to some other event. The first possibility corresponds to the case where one discontinuity forces another one; the second possibility describes the case of enabling. A simple example of a PTN constructed using the introduced primitives appears in fig. 1.

[1]This paper forms an entity together with [Mal91b] which describes in detail the full set of PTN primitives, and provides several examples of their possible usage.

Activities are a distinguished subtype of processes — they describe the behavior of some *agent*. *Actions* are activities which have "inherent termination conditions". Activity boxes are distinguished by doubling their vertical borders, and actions contain an additional internal box denoting their termination condition. In case of action there is no event following it — it is "encoded" in the termination conditions of the action. However, if one wants this event to enable some other event, then the enabling arrow is drawn directly from the box representing the termination condition.

Activities and actions are used to focus attention on processes ascribed to some agent, or agents. Later on we will assume that every PTN contains at least one cycle consisting solely of activities.

The PTN language allows one to combine enabling arrows using both **and** and **or** connectives. The **and** of two enablings is expressed by two consecutive events on one transition from a process to another process. The **or** may be represented with two different paths leading from one process to another, but is usually depicted by merging the enabling arrows themselves.

In the following example, PTNs have been used to conceptualize the scenario of passing an intersection. Details can be found in [Mal91a]. Figures 1 and 2 present two PTNs describing, respectively, a very general overview of passing an intersection, and part of a much more detailed analysis of this scenario. Below an intuitive description of the contents of the depicted PTNs is given.

When an intersection becomes visible (what corresponds to the transition "No intersection in sight" → "Intersection visible", the car switches from the "DriveFreely" activity to the "Approach" action. The "Approach" action consists of an appropriate slowing down (up to stopping, if needed), checking the intersection state in order to determine if crossing is possible, and preparing to choose the outcoming road segment (left, right, or straight). After concluding the "Approach" (ie. coming to the meeting point of the road segment we are on, and the intersection) the car switches to the "Pass" action, which concludes passing through the intersection and causes the intersection to vanish. After "Passing" intersection the car switches to the "DriveFreely" activity again, until it meets the next intersection.

It is worth pointing out that the only conceptual difference between the "intersection" cycle, and the "car activity" cycle is that we ascribe the latter to the behavior of some intelligent agent, whereas the former describes the agent's environment. We also note that causal dependencies point both ways, so changes within the environment can cause some behavior change of the agent, and vice versa.

The "Approach" — "Pullup" — "Stop" — "Pass" — "DriveFreely" path (cf. fig. 2) contains the processes (activities) describing behavior of the *own* car. They correspond to the changing value of some fluent, say "State-of-my-own-car", which, in its turn, corresponds to some slot in the state vector containing all the sensory input data.

Because we don't explicitly mark events, but only transitions between processes (thick arrows on the pictures), enabling arrows (thin ones) are drawn between transitions, and not explicitly named events. All the labels in fig. 2 denoting triggering events (called configurations) correspond to different processes in some other part of the PTN describing configurations of crossing cars. These labels should be read as follows:

A – no crossing cars;

RL[2] (RS, RR, FL, FS, FR, LL, LS, LR) – there is at least one car having the RL (RS, RR, FL, FS, FR, LL, LS, LR) property among the cars approaching the intersection from other directions than the OWN car does.

[2] In this one, and in the following two-letter labels, the first letter corresponds to the direction the crossing car comes from, the second letter — to the direction it intends to turn. R = right, L = left, S = straight.

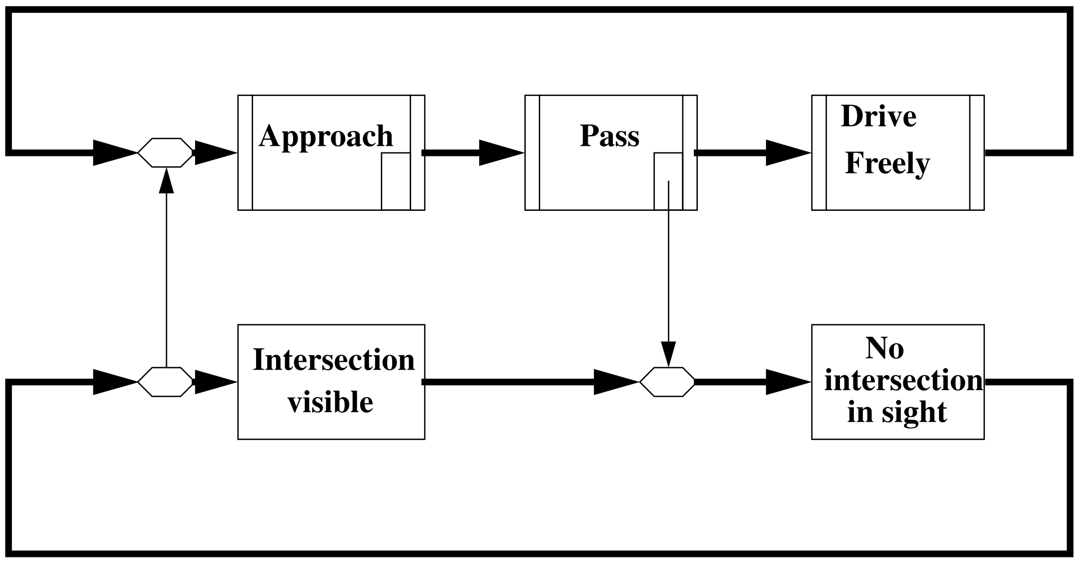

Figure 1: How to pass an intersection.

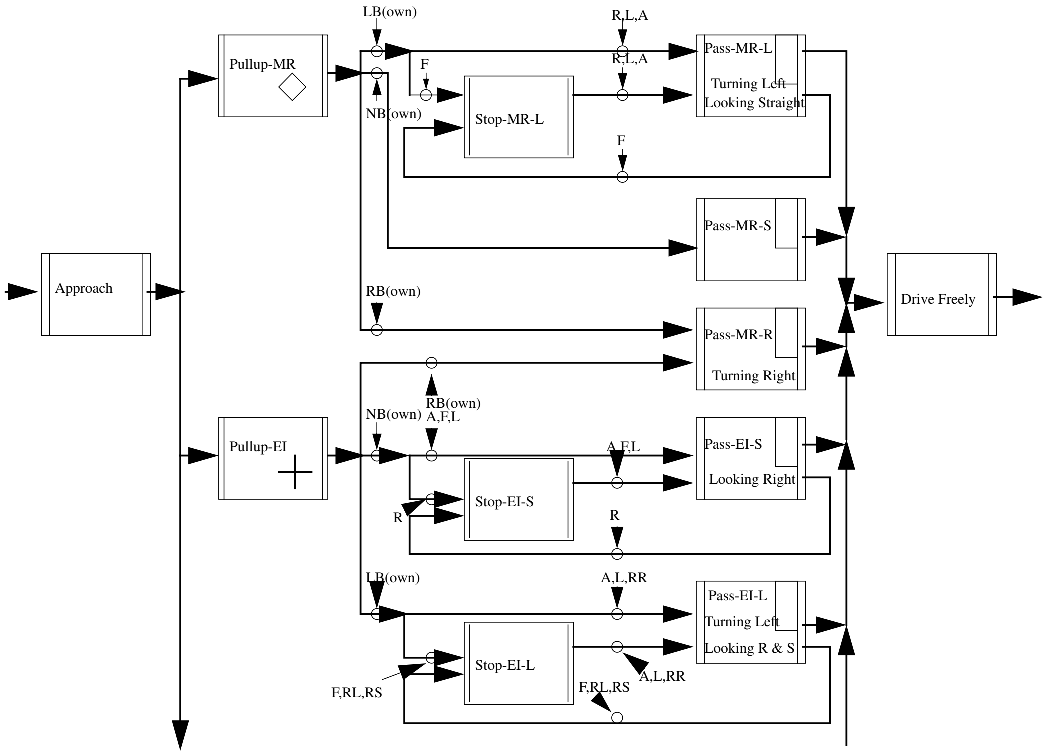

Figure 2: A fragment of "How to pass an intersection", with more details shown.

R (F, L) – there is at least one car with either an RL, RS, or RR (either FL, FS, or FR; either LL, LS, or LR) property among the cars approaching the intersection from other directions than the OWN car does.

It should be stressed that PTNs strongly encourage a hierarchical approach to the task of domain description. We can start with a very rough conceptualization, distinguishing only major properties of the world around us, and their most important values, and then we can successively refine this first approximation, obtaining an accurate (and, what is important in our case, ready to implement) description of (hopefully) all the important interactions among some agent(s) and its (their) environment. Figures 1 and 2 illustrate to some extent this top-down approach, where one box from fig. 1 is split into several ones in fig. 2.

Another advantage of PTNs is the following: they can be used to graphically present and explain the actual contents of a knowledge base. Although this factor is less relevant from a formal point of view, it becomes very important in cases where domain experts, not necessarily fully competent in formal logic, need to be involved in the process of creation, validation, and expansion of a large and complex knowledge base.

We now turn from the informal description of PTNs to the more formal aspects of the PTN language.

3 Definition

A *PTN* is a directed, labeled, finite graph $< N, A, L >$, where N is a finite set of *nodes*, A is a finite set of *arcs*, and L is a *labeling function*. $N = P \cup E, P \cap E = \emptyset$. P denotes a set of *processes*, E is a set of *events*. $P \supseteq A_v \supseteq A_n$, where A_v is a set of *activities*, and A_n a set of *actions*.

The *labeling function* $L : N \rightarrow F$ associates with every node a label (resembling a slot of a frame structure). $L : n \mapsto f \in Attr \times Val$, where $Attr$ is a finite set of *attribute names*, $Val = \bigcup_{a \in Attr} Va(a)$ is a set of all possible attribute values, and Va is a function assigning to every attribute name the set of all possible values for that name. $\forall_{a \in Attr} \overline{\overline{Va(a)}} < \aleph_0$, i.e. each set of attribute values should be finite.

$A \subset N \times N$, moreover $A = Ta \cup Ea$, $Ta \cap Ea = \emptyset$, where Ta is a set of *transition arrows*: $Ta \subset (P \times E) \cup (E \times P)$, and Ea is a set of *enabling arrows*: $Ea \subset E \times E$.

The set of transition arrows Ta fulfills the following constraints:

- Every event has at least one preceding process and at least one succeeding process.

$$\forall (n \in E) \exists (p \in P) < n, p > \in Ta \tag{1}$$

$$\forall (n \in E) \exists (p \in P) < p, n > \in Ta \tag{2}$$

- Every event can have at most one successor.

$$\forall (< n_1, n_2 > \in Ta)(n_1 \in E \Rightarrow \forall (< n_3, n_4 > \in Ta)(n_3 = n_1 \Rightarrow n_4 = n_2)) \tag{3}$$

- Every event can have at most one predecessor.

$$\forall (< n_1, n_2 > \in Ta)(n_2 \in E \Rightarrow \forall (< n_3, n_4 > \in Ta)(n_4 = n_2 \Rightarrow n_1 = n_3)) \tag{4}$$

The set Ea of enabling arrows consists of two subsets: $Ea = Epa \cup Eea$, $Epa \cap Eea = \emptyset$. Epa denotes the set of *proper* enablings (event enabling other event), and Eea denotes the set of *extended* enablings (both event and its succeeding process enable some event). Their difference will be explained in the next section, where we assign a semantics to them.

Let *transition* be a pair of transition arrows of the form $<< p_1, e >, < e, p_2 >>$, where $p_1, p_2 \in P$, and $e \in E$. From (1) — (4) it follows that for each pair p_1, p_2 if such an e exists then it is unique, so we can treat transitions as pairs from $P \times P$.

Let T denote the set of all transitions in a PTN. Let $t \in T$. $s(t)$ will denote the *source* of the transition t, i.e. $s(t) = s(< p_1, p_2 >) \stackrel{\mathrm{df}}{=} p_1$, and $d(t)$ will denote the *destination* of the transition t, i.e. $d(t) = d(< p_1, p_2 >) \stackrel{\mathrm{df}}{=} p_2$ (in other words, they are projections of t to the first and second axes, respectively).

$C = \{t_1, t_2, ..., t_n\}$ will be called a *chain* of transitions iff $\forall(i \in \{1, ..., n-1\}) d(t_i) = s(t_{i+1})$. A chain will be called *proper* iff $\forall(i, j \in \{1, ..., n\})(i \neq j) \Rightarrow d(t_i) \neq d(t_j)$, i.e. if it contains no cycles. Let $\mathcal{CP}$ denote the class of all proper chains.

The following constraints should be fulfilled by a PTN:

- Every process is a member of some cycle.

$$\forall(p \in P)\exists(c \in \mathcal{CP})((c = \{t_1, ..., t_n\}) \Rightarrow (p = s(t_1) \& p = d(t_n)) \tag{5}$$

- Every transition belongs to some cycle.

$$\begin{aligned} \forall(p, q \in P)\{\exists(c_1 \in \mathcal{CP})((c_1 = \{t_1, ..., t_n\}) \,\&\, (p = s(t_1) \& q = d(t_n)) \quad \Rightarrow \\ \Rightarrow \exists(c_2 \in \mathcal{CP})((c_2 = \{t'_1, ..., t'_m\}) \,\&\, (q = s(t'_1) \& p = d(t'_m))\} \end{aligned} \tag{6}$$

Let C_p, called the *cycle set* of p, denote the set of *all* chains fulfilling (5), i.e. the set of all cycles p is an element of.

- Every cycle consists of at least two transitions.

$$\forall(p \in P)\forall(c \in C_p) card(c) \geq 2 \tag{7}$$

where the function *card* yields the cardinality of its argument.

In order to write down the last two properties of PTNs, we introduce the notion of a *cycle group* of some process p (denoted by $CG(p)$). $CG(p)$ consists of all the processes connected with each other via some proper chain (cf. Fig. 3):

$$\begin{aligned} CG(p) \stackrel{df}{=} \{q \in P \mid \exists(c \in \mathcal{CP})((c = \{t_1, ..., t_n\}) \& [(p = s(t_1) \& q = d(t_n)) \quad \vee \\ \vee (q = s(t_1) \& p = d(t_n))])\} \end{aligned} \tag{8}$$

Alternatively, we can define $CG(p)$ recursively:

$$CG(p) \stackrel{df}{=} \bigcup_{c \in C_p} c \cup (\bigcup_{q \mathrel{\widetilde{\in}} c \in C_p} CG(q))$$

where

$$q \mathrel{\widetilde{\in}} c \stackrel{df}{\Leftrightarrow} c = \{t_1, ..., t_n\} \,\&\, \exists(i \in \{1, ..., n\})(d(t_i) = q)$$

Then we can formulate the following two requirements:

- Every cycle containing an activity consists only of activities.

$$\begin{aligned} \forall(c \in CG(p))\{[(c = \{t_1, ..., t_n\}) \& \exists(i \in \{1, ..., n\})(s(t_i) \in Av \vee d(t_i) \in Av)] \quad \Rightarrow \\ \Rightarrow \forall(i \in \{1, ..., n\})(s(t_i) \in Av \,\&\, d(t_i) \in Av)\} \end{aligned} \tag{9}$$

- Enabling arrows link only different cycle groups.

$$\begin{aligned} &\forall(e \in Ea)\forall(p_1, p_2, p_3, p_4 \in P) \\ &\quad((e =< e_1, e_2 > \,\&\, < p_1, e_1 >, < e_1, p_2 >, < p_3, e_2 >, < e_2, p_4 > \in Ta) \quad \Rightarrow \\ &\quad\quad \Rightarrow (C(p_1) = C(p_2) \& C(p_3) = C(p_4) \,\&\, C(p_1) \cap C(p_3) = \emptyset)) \end{aligned} \tag{10}$$

This concludes the definition of PTNs.

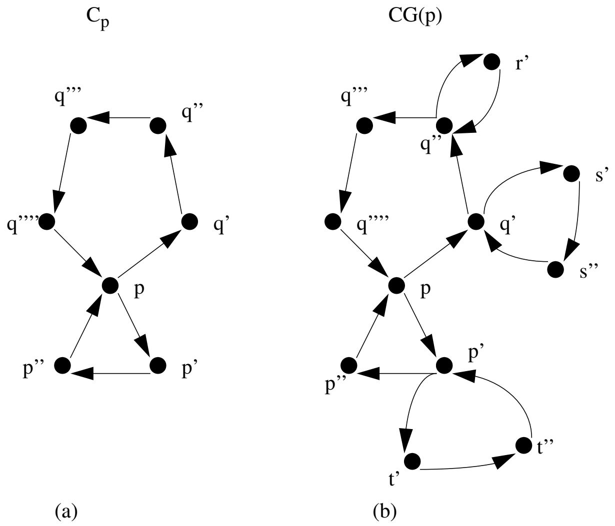

Figure 3: Cycle set, and cycle group of p.

4 Semantics

This section contains an informal description of PTN semantics (presented in [Mal92]). We provide a formal semantics in order to be able to compare PTNs with the family of Feature Logics [San94] which are the frame of reference for all the knowledge representation tools used within our project.

Our domain will consist of *temporal frames*, frames corresponding to the usual structures described in AI textbooks, with a relatively close correspondence to our ontology. (The world can be described using properties; values of properties are functions of continuous time.)

We begin with the formal definition of a *frame*. First we introduce a set T of *time points*. The only requirement is that it is totally ordered. A *frame* f is a finite set of pairs, which are called *slots*. A slots consist of an *attribute name* belonging to some predefined set, and a function assigning a value to the attribute.

By *domain*, we understand a finite set D of frames. Two functions are introduced: the first one describing the correspondence between slots and process cycles, and the second one which assigns slot values to particular processes within the cycle. First, we introduce a *correspondence function* Q which assigns to every process of the PTN a triple <attribute-name, value-function, set-of-truth-valued-functions>, where the pair <attribute-name, value-function> is a slot of some frame from the domain. We say that a correspondence function Q is *regular* iff every element of the set-of-truth-valued-functions component of Q has the same value for all time points for which the slot-value component gives the same value and, moreover, uniquely distinguishes (yielding the value $\mathcal{T}$) one particular value of this slot.

Then we introduce the *assignment function* R which assigns to every process of the PTN a pair consisting of some value of the corresponding slot (as established by Q) and an appropriate truth-valued function of time distinguishing this value (by yielding value $\mathcal{T}$) from others.

An *interpretation* I is then a tuple $< T, D, Q, R >$, T being a set of time points, D a domain, Q a regular correspondence function, and R an assignment function.

The semantics for PTNs is defined using the components introduced above. Normally, we are interested in an intended model and therefore the satisfiability of a PTN in that model is the predominant concern. It is difficult to imagine a "tautological" PTN, satisfied in all possible interpretations, so we will concentrate on satisifiability.

A PTN $< N, A, L >$ is *satisfied* by some interpretation $< T, D, Q, R >$ iff

1. Every process in a cycle group corresponds to the same slot.
2. The "slot value" function is piecewise constant, and the image of an attribute through this function is a finite set.
3. Different processes in a cycle group correspond to different values of the appropriate slot.
4. Different values of a slot correspond to different processes in a cycle group.
5. For every change of a slot-value function from w_1 to w_2, there exists a transition between processes corresponding to w_1 and w_2.
6. For every enabling arrow, the appropriate dependence between value functions holds.
7. For every process in a cycle, the corresponding slot has the same name and one of the admissible values.
8. There exists precisely one process visited (i.e. yielding value $\mathcal{T}$ of the appropriate component of the assignment function R) in a cycle group at a time.

5 PTNs vs. Petri nets

To show that any PTN can be reformulated as a Petri net (PN) we will sketch a translation procedure yielding, for a given PTN, a PN having the same behavior (or meaning) as the one ascribed to the original PTN.

More complex PTNs (using the full set of language possibilities, see e.g. [San90]) can be rewritten without change of their meaning using only those elements which have already been defined in the section 2. These elements are: processes (and also activities and actions as their special cases) and events as "building blocks", and transition and enabling arrows as links between these blocks. Therefore we only need to show that these basic elements can be translated into PNs in the desired way.

We use the "conditions and events" terminology of Petri nets theory as it directly corresponds to our intuition, however, the pictorial representation used here is the most popular one (see e.g. [Pet81], and is often associated with the "places and transitions" terminology.

Any PTN process is assigned a corresponding PN condition (or state, or place), and every PTN event has one corresponding PN event. PTN transition arrows are also mapped one-to-one into PN arcs pointing from conditions to events, or vice versa. So, for example, a simple PTN as shown in fig. 4, would be translated into the PN shown in fig. 5.

It remains to be shown how the PTN's enabling arrows can be translated into the PN's dependencies. The following figures present, respectively, a simple PTN with proper enablings only (fig. 6), and its translation into a PN (fig. 7). The next set of figures (figs. 8, and 9) presents a translation of extended enablings (ie. enabling arrows coming both from an event, and from subsequent process). The translation procedure itself is obvious and will not be formulated here. It can be proved that a PTN, and the PN given by the procedure described above, are behaviorally equivalent.

6 Conclusions

In this paper we have presented a formal definition of Process Transition Networks — a pictorial language suitable for the conceptualization, and coding of knowledge about complex, quantitatively intractable environments. The word "coding" is justified by the fact that sentences in this language (i.e. PTNs) correspond (can be mechanically translated) to a very restricted temporal logic language RETFL which we are using in our project [San94]. The PTN language

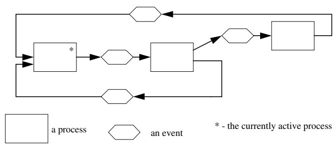

Figure 4: A very simple PTN (with no enabling arrows).

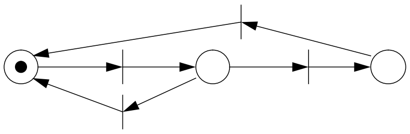

Figure 5: Corresponding Petri Net.

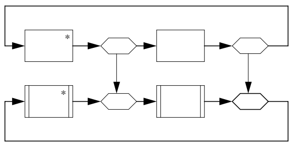

Figure 6: A very simple PTN (with proper enabling arrows).

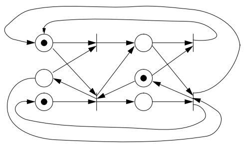

Figure 7: Petri net corresponding to fig. 6.

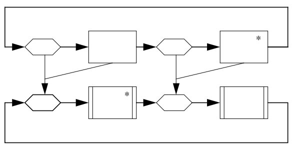

Figure 8: A simple PTN (with extended enabling arrows).

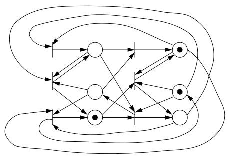

Figure 9: Petri net corresponding to fig. 8.

is equivalent to a subset of classical Petri nets formalism. This suggests one obvious direction of further study: what properties formulated in the Petri nets framework hold for PTNs.

The representational advantages of the language are simplicity, possibility of expressing causal dependencies, ability to describe the behavior of several agents acting in the same environment, modularity, and the possibility of using it for subsequent refinements of an initial conceptualization. These properties have been confirmed during the conceptualization of several non-trivial traffic-behavior scenarios.

One obvious disadvantage is the fact that we introduce yet another language for expressing knowledge about the real world, moreover a very restricted one. However, none of the existing formalisms is suitable for our purpose, that is modelling complex dynamic environments, either because of much generality, or because of lack of expressivity. One possible alternative would be to use the statechart formalism [Har87], however its set of automata-theory-based primitives does not quite fit properly to our needs. In our opinion, the communicativity of PTN language, and its ease of use, justifies its introduction.

Acknowledgment

Discussions with Simin Nadjm-Tehrani and Erik Sandewall influenced the contents of the paper. Jacek Wrzos-Kamiński, Hua Shu, Patrick Doherty, and anonymous reviewers helped to improve readability.

References

[Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[Mal91a] Jacek Malec. How to pass an intersection. In *Proc. of the Prometheus Pro-Art Workshop on Intelligent Co-Pilot*, pages 167–182, Grenoble, France, 1991.

[Mal91b] Jacek Malec. Process Transition Networks: What Are They For? In *Proceedings of the IEEE Systems, Man and Cybernetics Conference, Charlottesville, VA*, pages 1177–1182, October 1991.

[Mal92] Jacek Malec. Process Transition Networks: The Final Report. Research Report LiTH-IDA-R-92-07, Department of Computer Science, Linköping University, 1992.

[Pet81] James L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[San90] Erik Sandewall. Proposal for a ProArt specification platform. Technical Report LAIC-IDA-90-TR18, IDA, Linköping University, 1990.

[San94] Erik Sandewall. *Features and Fluents.* Oxford University Press, 1994.