Compositional Design and Verification of Communication Protocols, using Labelled Petri Nets

JEAN CHRISTOPHE LLORET
VERILOG, 150 rue Nicolas Vauquelin, 31081 Toulouse cedex,
PIERRE AZÉMA, FRANÇOIS VERNADAT
LAAS-CNRS, 7 Avenue Colonel Roche, F-31077 Toulouse cedex

1 Introduction

This paper proposes a methodology to specify and verify telecommunication protocols by means of Labelled Predicate Transition nets (LPrT). In this paper, the following two principles are used as guidelines.

Modular Specification The structured system decomposition is based upon communication primitives. The rendez-vous communication paradigm of ISO language LOTOS is extended to multi-gate rendez-vous. Several input/output events may appear on a single transition: Petri Net transitions are labelled by sets of communicating events.

Incremental Description A single system is described with respect to several levels of abstraction. Each new abstraction level supplies a more detailed model and new properties are to be verified. Communication by multi-rendez-vous is a first means of abstraction. A second means concerns the data part. To facilitate data abstraction in a LPrT net, logic programming (Prolog) is used as a declarative and prototyping language.

The main contributions of this paper concern the support of former principles. Multi-rendez-vous is introduced in a stepwise approach, from basic semantical models, that is Labelled transition systems (LTS) and Labelled Petri Nets (LPN), to labelled Predicate Transition nets. The labelled Predicate Transition nets provide the highest abstraction level and are the user interface model. This stepwise definition presents the following characteristics:

didactical: Synchronization aspects are studied in the context of labelled Petri Nets. Communication with value passing is only introduced for Labelled Predicate transition nets.

analytical: Verification techniques are based on the analysis of the LPrT model behaviour. A specific technique, so-called projection or service computation, is derived from observational equivalence [Mil80]). The global service results from the composition of sub-net services. A modular design of labelled PrT nets entails a modular verification.

The composition of Labelled Petri Nets is described in Section 2. A value passing mechanism and the parameterization of Petri Nets are illustrated in Section 3 by means of a specific application.

2 Composition of multi-event Labelled Petri Nets

This section introduces Labelled Petri Nets and multi-rendez-vous operator $|_{LPN}$ as the core language (no value passing) and the basic composition operator, respectively.

The expressive power of this composition operator results from the use of multi-events actions which enable to specify rendez-vous among several (more than two) transitions, on the same or distinct interaction points. This multiple-rendez-vous is an abstraction with respect to implemented communication mechanisms. In the context of the advocated progressive modelling, multi-rendez-vous enables the design of abstract and very compact models.

2.1 Multi-events actions

Communication actions are defined according to the following principles:
events: An event is the most elementary communication unit. Let α be the set of gates, let V be the set of interactions. An event is a couple (g, v) which consists of gate g and interaction v. Event (g, v) is denoted $g(v)$. Let *gate* be the function which returns the gate of an event ($gate(g(v)) = g$) and let $\mathcal{E}$ be the set of events.
1. Multi-event action: *an action describes several communication events that are performed synchronously on different gates.* A transition is labelled by an action. Because an action refers to a set of events, the expressiveness is increased with respect to the reference languages CCS [Mil80] and LOTOS in which an action is either a single (observable) event or internal action i.
2. A single event per gate that is gate is an unshared resource. A service access point is dedicated to a single entity. Consequently, *two distinct events on the same gate do not belong to the same action.*
Formally, an action A is a subset of events ($A \subseteq \mathcal{E}$) such that,
$\forall e_1, e_2 \in A, e_1 \neq e_2 \Rightarrow gate(e_1) \neq gate(e_2)$.

Action $\emptyset$ is denoted i (no event = internal action).

Definition 2.1 Labelled Transition system labelling

Let *proc* be a labelled transition system; let $\mathcal{T}$ be *proc* transition set. Let $\alpha_{proc} \subseteq \alpha$ be a subset of gates, let V be the set of interactions.
An event e, connected to a gate in α_{proc}, is a couple $(gate, interaction) \subset \alpha_{proc} \times V$. e is denoted $gate(interaction)$.
proc labelling is couple $(\alpha_{proc}, l_{proc})$ where l_{proc} is the label function. The domain of label function l_{proc} is transition set $\mathcal{T}$. The range of function l is the subset of actions constituted by events connected in α_{proc}.
In the sequel, $\mathcal{L}_{proc}$ denotes couple $(\alpha_{proc}, l_{proc})$.

2.2 Multi-rendez-vous Composition of Labelled Transition Systems

Labelled Transition Systems supply an operational semantics to Labelled Petri Nets. The multi-rendez-vous of labelled transition systems is the interpretation of multi-rendez-vous of Petri nets.

Definition 2.2 *Labelled Transition System*

A labelled transition system *proc* is a 5-tuple $(S, \mathcal{T}, -t \rightarrow_{t \in \mathcal{T}}, s_0, \mathcal{L})$ where:

- S set of states.
- T set of transitions.
- $-t \rightarrow \subset S \times S$ state change performed by transition t.
- $s_0 \in S$, initial state.
- $\mathcal{L}$ is a labelling as defined above.

Two labelled Transition systems $proc_1$ and $proc_2$ are composed relative to their common gates.

LTS common gates: the set of common gates $\alpha_{proc_1} \cap \alpha_{proc_2}$ is denoted $\alpha_\cap$.

events to be synchronized: events to be synchronized of label $l(t)$ are events whose gate is a shared gate between $proc_1$ and $proc_2$. Notation is $sync_{\alpha_\cap}(l(t))$:

$sync_{\alpha_\cap}(l(t)) = \{e \in l(t) \mid gate(e) \in \alpha_\cap\}$.

Definition 2.3 *LTS composition operator* $|_{LTS}$ Let $proc_i = (S_i, \mathcal{T}_i, -t_i \rightarrow_{t_i \in \mathcal{T}_i}, s_{i,0}, \mathcal{L}_i)_{i=1,2}$ be two LTS.

Composed LTS $proc_1 \,|_{LTS}\, proc_2$ is $(S_1 \mid S_2, \mathcal{T}_1 \mid \mathcal{T}_2, -t \rightarrow_{t \in \mathcal{T}_1 | \mathcal{T}_2}, s_{1,0} \mid s_{2,0}, \mathcal{L}_1 \mid \mathcal{L}_2)$ where:

$\mathcal{L}_1 \mid \mathcal{L}_2$ is $proc_1 \mid proc_2$ labelling defined by set of gates $(\alpha_{proc_1} \cup \alpha_{proc_2})$ and labelling function l defined on domain $\mathcal{T}_1 \mid \mathcal{T}_2$.

Function l is defined together with composed states and composed transitions by the following derivation rules (s_i, s_i' and t_i are states and a transition of $proc_i$ respectively):

Independant execution

$$(1)\ \frac{s_1 - t_1 : l_1 \rightarrow s_1'}{s_1 \mid s_2 - t_1 : l_1 \rightarrow s_1' \mid s_2} \quad (sync_{\alpha_\cap}(l_1) = \emptyset)$$

$$(2)\ \frac{s_2 - t_2 : l_2 \rightarrow s_2'}{s_1 \mid s_2 - t_2 : l_2 \rightarrow s_1 \mid s_2'} \quad (sync_{\alpha_\cap}(l_2) = \emptyset)$$

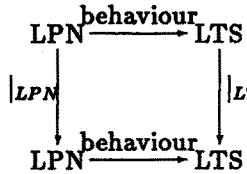
Synchronization

$$(3)\ \frac{s_1 - t_1 : l_1 \rightarrow s_1',\ s_2 - t_2 : l_2 \rightarrow s_2'}{s_1 \mid s_2 - t_1 \mid t_2 : (l_1 \cup l_2) \rightarrow s_1' \mid s_2'} \quad (sync_{\alpha_\cap}(l_1) = sync_{\alpha_\cap}(l_2))$$

2.3 Labelled Petri Nets

Firstly, Labelled Petri Nets are defined as Place/Transition nets of capacity one, associated with a labelling as defined above.

The behaviour of a labelled Petri Nets is then introduced as a labelled transition system. The marking graph and the step graph are two possible candidates for describing a net behaviour. In the step graph, parallel executions are explicitly taken into account and thus a more precise representation of the net behaviour is given. Labelled Petri Net operator $|_{LPN}$ is interpreted on step graphs by Labelled Transition System operator $|_{LTS}$ in such a way that diagram 1 commutes.



LPN: Labelled Petri Nets
LTS: Labelled Transition Systems
$\mathcal{B}(N)$: net N behaviour (step graph)

$\forall N_1, N_2 \in LPN, \mathcal{B}(N_1 \mid_{LPN} N_2) = \mathcal{B}(N_1) \mid_{LTS} \mathcal{B}(N_2)$

Figure 1: Semantics of Labelled Petri Nets composition "$|_{LPN}$"

Definition 2.4 *Labelled Petri Net*

A labelled Petri Net is a 6-tuple $N = (P, \mathcal{T}, pre, post, M_0, \mathcal{L}_N)$ where:

- P is a set of places;
- $\mathcal{T}$ is a set of transitions, $P \cap \mathcal{T} = \emptyset$
- $pre, post : \mathcal{T} \rightarrow P(P)$ are two mappings which connect a transition to two sets of places called preconditions and postconditions respectively.
- M_0 is a set of places called initial marking.
- $\mathcal{L}_N$ is the net labelling defined by couple (α_N, l_N) as introduced previously.

Example: net N_1, Fig 2, depicts concurrent transitions t_1 and t_2 (places are circles, transition boxes inscribed by a transition name; if a transition box is connected to a gate, the edge is annoted by the value of the corresponding transition event; the preconditions and postconditions of a transition are the input and output places respectively; places with a token belong to the initial marking).

Definition 2.5 *Firing rule* The transition firing rule proposed for "augmented condition/event nets" in [PD87] is adopted. But the definition of parallel transitions differs because an interaction point is a non shared resource: with respect to labelled Petri nets, two transitions connected to a common interaction point cannot be fired in parallel.

Let M be a marking (or set of places) and T a set of transitions (set T is called *step*). T is M enabled (notation $M[T >$) iff

- T transitions are *pairwise independent* that is two distinct transitions $t_1, t_2 \in T$ share neither a place (*place independance*: $(pre(t_1) \cup post(t_1)) \cap (pre(t_2) \cup post(t_2)) = \emptyset$) nor a gate (*gate independance*: $gate(l_N(t_1)) \cap gate(l_N(t_2)) = \emptyset$, where $gate(l_N(t)) = \{gate(e) \mid e \in l_N(t)\}$).
- preconditions of T transitions are fulfilled ($pre(T) \subset M$, with $pre(T) = \bigcup_{t \in T} pre(t)$) and postconditions which are not preconditions are not fulfilled ($M \cap (post(T) \setminus pre(T)) = \emptyset$).

Marking M' results from firing step T in marking M (notation $M[T > M'$) iff step T is enabled in M and $M' = (M \setminus pre(T)) \cup post(T)$.

The behaviour of a LPN is a LTS such that a state is a reachable marking and a LTS transition is a firable step.

Definition 2.6 *LPN Behaviour*

Let $N = (P, \mathcal{T}, pre, post, M_0, \mathcal{L}_N)$ be a LPN. Reachable markings and firable steps of net N are respectively the smaller sets $M_0[>$ and $M_0[$ defined by:

- $M_0 \in [M_0 >$
- If $M \in [M_0 >$ and T enabled in M with $M[T > M'$ then $M' \in M_0[>$ and $T \in M_0[$.

Net behaviour is LTS, $\mathcal{B}(N) = (M_0[>, M_0[, -T \rightarrow_{T \in M_0[}, M_0, \mathcal{L}_\mathcal{B})$ where:
• change of state in $\mathcal{B}(N)$ corresponds to step execution:$M, M' \in M_0[>, M - T \rightarrow M' \Leftrightarrow M[T > M'$.
• Behaviour labelling $\mathcal{L}_\mathcal{B} = (\alpha_B, l_B)$ directly follows net labelling: the gate set of $\mathcal{L}_\mathcal{B}$ is $\mathcal{L}_N$ one, i.e. $\alpha_\mathcal{B} = \alpha_N$. Labelling function $l_\mathcal{B}$ is the canonical extension to sets of the net transition labelling function l_N: $\forall T \in M_0[, l_\mathcal{B}(T) = \bigcup_{t \in T} l_N(t)$.

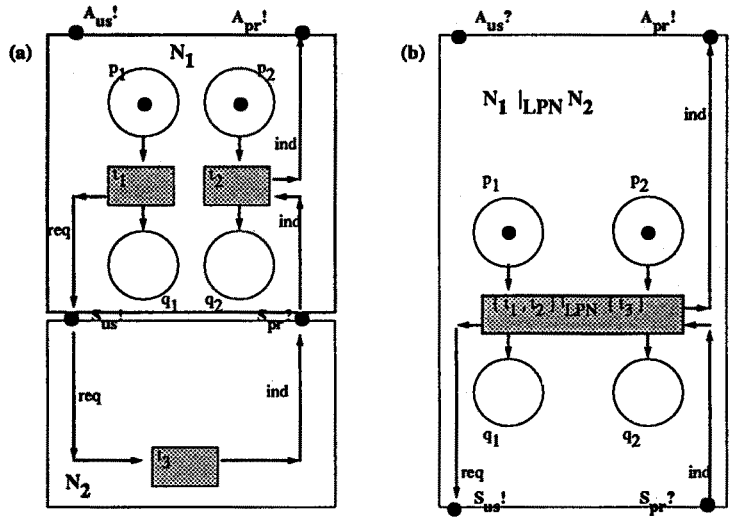


Figure 2: Parallel composition of Labelled Petri Nets

2.4 Multi-rendez-vous Composition of Labelled Petri Nets

Operator $|_{LPN}$ is defined in order to simulate composition of behaviours. The three following characteristics of net multi-rendez-vous are worth noticing:
1. *Parallelism between transitions is taken into account* When composing two labelled Petri nets, two transitions of a net, which fire in parallel, may be merged with a transition of the other net (on Fig 2 transition $\{t_1, t_2\}$ $|_{LPN}$ $\{t_3\}$ results from the merging of transition sets $\{t_1, t_2\}$ and $\{t_3\}$).
2. *Minimal mergings* The possibility of running synchronously (transitions are in parallel) is distinguished from a necessary synchronization (transitions are merged). In the composition of two nets, all synchronization possibilities are preserved but the number of mergings is minimal.
3. The derivation of a composed net N_1 $|_{LPN}$ N_2 does not require computation of behaviours $\mathcal{B}(N_1)$ and $\mathcal{B}(N_2)$.

Let $N_i = (P_i, T_i, pre_i, post_i, M_{0,i}, \mathcal{L}_i)_{i=1,2}$ be two LPN with labelling $\mathcal{L}_i = (\alpha_i, l_i)_{i=1,2}$ and let $\alpha_\cap$ be the common gate alphabet.
Preliminary definitions:
Transition to be merged: Transition t of net N_1 or N_2 is to be merged iff there is at least one event to be synchronized in t label (recall that an event to be synchronized is

an event whose gate belongs to common gate set $\alpha_\cap$).

Mergeable Transition sets: Two transition sets $T_1 \subseteq \mathcal{T}_1$ and $T_2 \subseteq \mathcal{T}_2$ are mergeable iff T_1 (resp. T_2) is a set of transitions *to be merged*, pairwise *independent* and the set of events to be synchronized of T_1 transitions equals the one of T_2 transitions (i.e. $sync_{\alpha_\cap}(l_1(T_1)) = sync_{\alpha_\cap}(l_2(T_2))$).

Minimal mergeable Transition sets: T_1 and T_2 are minimal mergeable transition sets iff T_1 are mergeable transition sets and if it is not possible to partition T_1, T_2 merging in smaller ones that is: for all $T_1' \subset T_1$ and $T_2' \subset T_2$, T_1' and T_2' are not mergeable.

Definition 2.7 *LPN composition operator* $|_{LPN}$

Composed net $N_1 \,|_{LPN}\, N_2$ is $(P_1 \cup P_2, \mathcal{T}_1 \,|_{LPN}\, \mathcal{T}_2, pre, post, M_{1,0} \cup M_{2,0}, \mathcal{L}_1 \,|_{LPN}\, \mathcal{L}_2)$ where:

- $\mathcal{T}_1 \,|_{LPN}\, \mathcal{T}_2$ transitions are, on the one hand, $\mathcal{T}_1$ and $\mathcal{T}_2$ transitions which are not to be merged and, on the other, merged transitions $T_1 \,|_{LPN}\, T_2$ where T_1, T_2 are minimal mergeable transition sets.
- Preconditions and postconditions of a nonmerged transition are unchanged in composed net. Pre and postconditions of merged transition $T_1 \,|_{LPN}\, T_2$ are defined as the set union of the respective pre and postconditions: $pre(T_1 \,|_{LPN}\, T_2) = pre_1(T_1) \cup pre_2(T_2)$ and similarly for postconditions.
- New labelling $\mathcal{L}_1 \,|_{LPN}\, \mathcal{L}_2$: gate sets of the composed nets are added; the labelling of a nonmerged transition is unchanged; labelling of merged transition $T_1 \,|_{LPN}\, T_2$ is $l_1(T_1) \cup l_2(T_2)$.

Example: figure 2b depicts composed net $N_1 \,|_{LPN}\, N_2$.

Proposition 2.1 *Properties of multi-rendez-vous composition*

Operators $|_{LTS}$ and $|_{LPN}$ are commutative and associative (up to state and transition bijective renaming).

The behaviour of a composed net is the composition of behaviours: $\mathcal{B}(N_1 \,|_{LPN}\, N_2) = \mathcal{B}(N_1) \,|_{LTS}\, \mathcal{B}(N_2)$ (up to state and transition bijective renaming).

3 Application

This section introduces Labelled Predicate Nets by means of an example. LPrT nets are a parameterized version of LPN; they are also PrT nets [Gen88] extended with labels, and featuring, in particular, direct execution in Prolog.

A remote reading mechanism, the so-called Telereport [HRJ89] application layer, is first modelled then analysed.

3.1 Models

Application entities A_{us} and A_{pr} cooperate through a nonperfect session service S (see Fig 3) in order to provide a remote reading service to a user process.

Application user service. The interface between the user process and entity A_{us} is of particular interest since it determines the external service provided to the user process: user request $req(C)$ is parameterized by requested data code C and is issued on Service Access Point, $A_{us}?$; confirmation $conf(Mess)$ is issued on SAP $A_{us}!$ and can

be of 3 types: *Mess* value is either "*error*": transmission by session service has failed, or "*nak*": no data of code C is available to the provider process, or C read value Val_C. Entity A_{us} is in charge of recovery if a transmission error occurs in the session service. Let NR_{Max} be the maximum number of consecutive recoveries with respect to the same read request. To study the correctness of the recovery mechanism a specific model of session service has been designed: session service may lose consecutively at most NE_{Max} messages. The provided service depends on relative values of parameters NR_{Max} and NE_{Max}.

Configuration. Three configuration classes are distinguished: (1) Perfect session service (no error $NE_{max} = 0$); (2) Faulty session service but less errors than recoveries ($NE_{max} \leq NR_{max}$); (3) Unreliable service: more errors than recoveries ($NE_{max} > NR_{max}$). Furthermore, the requested code may either be available to the provider (expected confirmation with value) or not (expected confirmation: "*nak*"). The set of potential configurations, and the database facts are the following:

	$conf_0$	$conf_1$	$conf_2$	*database*
NE_{Max}	0	succ(0)	succ(succ(0))	$MaxErr(NE_{max})$.
NR_{max}	succ(0)			$MaxRecover(succ(0))$.
code c_1	c_1 available $\rightarrow$ value v_1			$codeType(c_1)$. $codeVal(c_1, v_1)$.
code c_2	c_2 not available $\rightarrow$ *nak*			$codeType(c_2)$.

Session Service Net (see Fig 3, net S)
A request $req(C)$ or a response $resp(Mess)$ may be conveyed without error from user entity A_{us} to provider entity A_{pr} respectively and vice-versa. Primitive *req* becomes *ind* and *resp* becomes *conf*. The corresponding normal transitions are *transmitReq* and *transmitResp*. Request $req(C)$ or response $resp(Mess)$ may either be lost or incorrectly transmitted. This corresponds to transitions *loseReq* and *loseResp*. The number of consecutive errors before a reset signal (transition *reset*) is saved by variable K and bounded by logical condition $infMaxErr(K)$. Predicate $infMaxErr(K)$ is defined by the following clause:

```
/* infMaxErr(K) is true if K < NE_max */
infMaxErr(K) :- maxErr(NE_max),   /* database fact */
                inf(K, NE_Max).   /* true if K < NE_max */
```

The occurrence of an error results in sending message $conf(error)$ to user entity A_{us} (transition *error*).

User entity of Application protocol (see Fig 3, net A_{us})
The normal transition sequence is (1) *init* receipt of user process request $req(C)$, (2) *send* request transmission to session service, (3) *end* confirmation $conf(Mess)$ is received and returned to the user process, if $Mess \neq error$, that is *Mess* is a code value or "*nak*"). When a session service error is detected, via primitive $conf(error)$), recovery may take place, i.e. user process request is repeated. Transition *recover* is enabled if less than NR_{max} errors have occured; this enabling condition, $infMaxRecover(K)$, is defined by the following clause:

```
infMaxRecover(K) :- MaxRecover(NR_max)
                    inf(K, NR_max).
```

In case of confirmation $conf(error)$ and if the maximum number of recoveries is exceeded, transition *end* fires. Signal *reset* is sent to session service.

Provider entity of Application protocol (see Fig 3, net A_{pr})
Provider entity is initially ready to receive read indication $ind(C)$: transition *indication* is enable; procedure $readCode(C, ValOrNak)$ of transition *response* performs a read action; when C is available ($codeVal(C, Val)$ in database) the value of Val is substituted for variable $ValOrNak$; if C is not available variable $ValOrNak$ takes value *nak*, negative acknowledge. Procedure *readCode* is defined by the following net clause:

$readCode(C, Val) :-$	$codeVal(C, Val).$	code available
$readCode(C, nak) :-$	$\backslash + codeVal(C, Val).$	code not available

3.2 Verification

An "easy to use" verification technique is to compute the service provided by the protocol and to compare it with the expected one.

The global service provided by a behaviour (LTS) is a reduced LTS, minimal with respect to the state number, which preserves properties of observed communication actions; with respect to gate set G, a local service is derived from the behaviour after the hiding of events which occur on gates outside G. The standard observational equivalence [Mil80] is used to compute the service; observable actions sequences and deadlocks are preserved.

Service Derivation
Let N be a LPN system composed of three sub-nets: $N = N_1 \mid_{LPN} N_2 \mid_{LPN} N_3$. Net service $S(N)$ is defined as the service provided by the behaviour of N, i.e. $S(\mathcal{B}_{LPN}(N))$. There are two basic ways to compute service $S(N)$; a third one may also be derived in a combined manner:
Service composition: individual subnet services are computed first, then they are composed. We have, $S(N) \approx S(N_1) \mid_{LTS} S(N_2) \mid_{LTS} S(N_3)$. Partial verifications may be conducted on individual services, and then reused for the verification of the global service. Furthermore, as services are reduced behaviours, composition of individual services can be more efficient than computation of the global netbehaviour.
LPN composition sub-nets are composed to obtain a global net whose behaviour and service are derived in a second step. This approach is mandatory when individual behaviours are unbounded, even if the global behaviour is bounded. For example, in case of PrT nets, unbounded net may be a fifo queue model.
Combined approach: The former computations produce an equivalent result because diagram 1 commutes and because Milner observational equivalence $\approx$ is a congruence with respect to $\mid_{LTS}$. A combined approach may still be useful to gain a better insight into protocols. For example, $S(N) \approx S(N_1 \mid_{LPN} N_2) \mid_{LTS} S(N_3)$: on the one hand N_3 individual service is computed, and on the other net composition $N_1 \mid_{LPN} N_2$ may be performed.

Telereport Service Local service provided to user process is depicted in Fig 4 (transitions represent application service primitives). As long as errors are relatively few, i.e.,

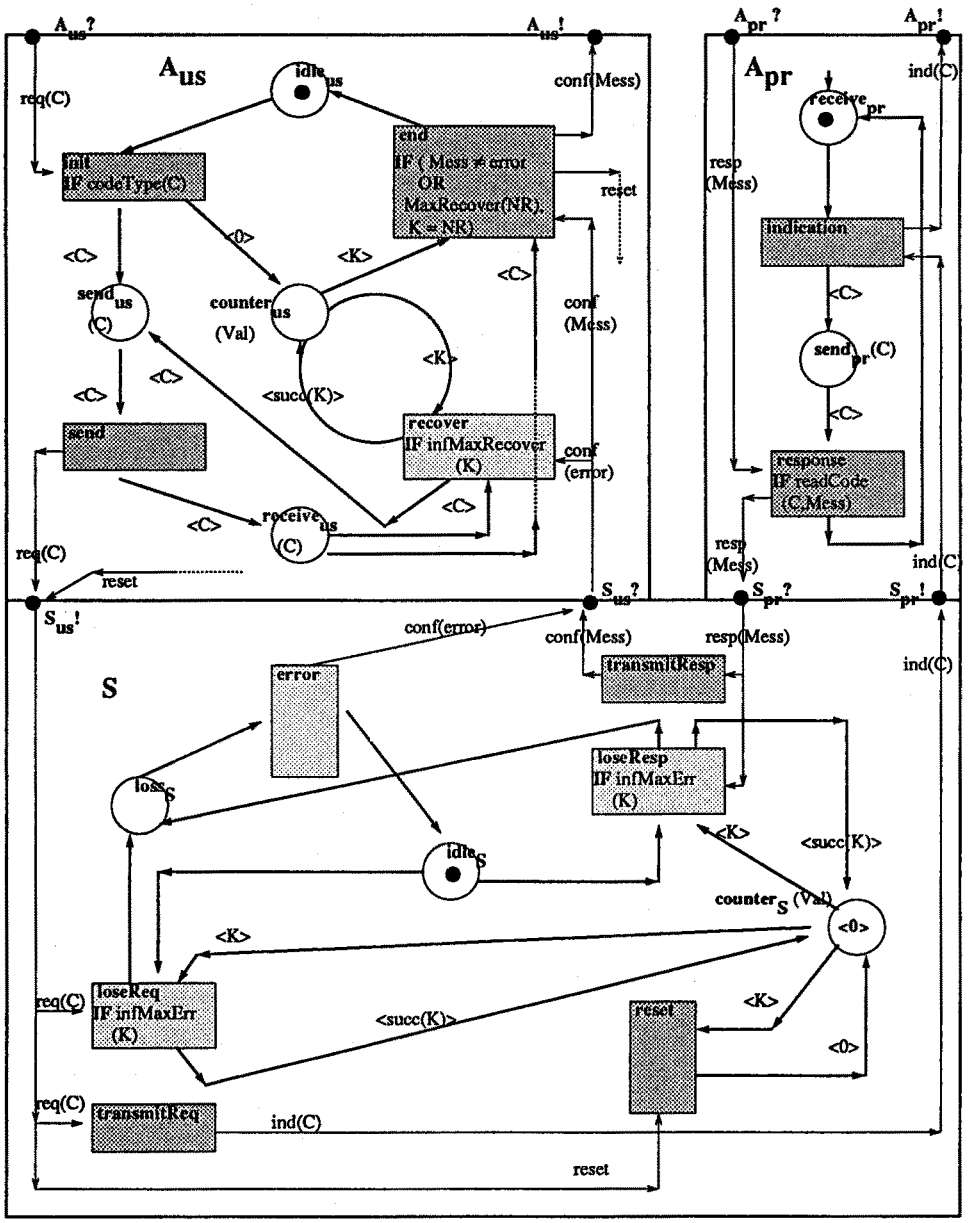


Figure 3: LPrT nets of Application entities A_{us} and A_{pr} and session service S

in configurations $conf_0$, $conf_1$, the provided service is error-free: the recovery mechanism plays the intended role. Configuration $conf_3$ is a degraded configuration: errors may be too numerous, and the application service may not deliver the expected answer (v_1 or *nak*). Correct status *error* is however delivered to user process. Confirmation $conf(error)$ follows an internal transition after the last (unrecovered) error.

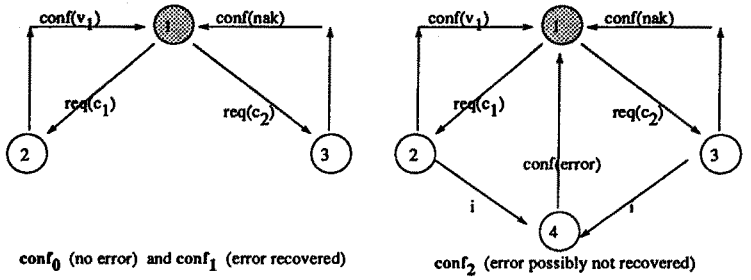


Figure 4: Application service with respect to user process

4 Conclusion

The introduced composition and verification techniques are implemented in the available software PIPN. The tool enforces structured and incremental approaches. It includes a graphical SADT-like editor that defines system architecture. The compiler computes global nets, as a result of subnet composition. State space can be interactively explored by means of the simulator. Finally, the complete behaviour can be computed and then reduced to protocol service. A model checker for CTL logic is also available. A current issue is to combine logic and service verification in a combined approach [PA89].

References

[Gen88] Hartmann J. Genrich. Equivalence transformation of PrT-nets. In *Workshop on Application and Theory of Petri Nets*, pages 229–248, 1988.

[HRJ89] P. Haudebourg, G. Revelaud, and T. Journey. Spécification du protocole de téléreport. Technical Report HR/23-2077 and HR23-1943, Electricité De France, DER, Clamart, 1989.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[PA89] F. Vernadat P. Azema, J.C. Lloret. Requirement analysis for communication protocols. In *Workshop On Automatic Verification Methods for Finite State Systems, Grenoble-France*, 1989.

[PD87] U. Montanari P. Degano, R. De Nicola. A distributed operational semantics fot ccs based on condition/event systems. Technical Report 111, Instituto di Elaboratione dell 'Informazione, C.C.R, Pisa, 1987.