On compositionality and Petri nets in protocol engineering[1]

N.A.Anisimov
Inst. for Automation and Control Processes, Russian Academy of Sci.
5 Radio Str., Vladivostok, 690041 Russia, anisimov@iapu2.marine.su

M.Koutny
Department of Computing Science, University of Newcastle upon Tyne
Newcastle upon Tyne NE1 7RU, U.K., Maciej.Koutny@newcastle.ac.uk

Abstract

This paper addresses the problem of designing communication protocols within the Petri net approach. Some recent results in combining Petri nets and compositionality are presented, and we argue that it should be possible to exploit these for protocol engineering. We outline a systematic approach to the design of protocol systems. At the top level, we use Petri net entities together with a set of operations. The external behaviour of entities is characterised using the notion of a bisimulation equivalence. At a lower level of design, we show how entities can be constructed from protocol procedures using suitable composition rules. The relationship between syntactic and behavioural notions of compositionality is also discussed.

Keywords

Communication protocols, protocol engineering, Petri nets, compositionality, protocol hierarchy, bisimulation equivalence

1 INTRODUCTION

From a historical perspective, the application of Petri nets to communication protocols dates back to the very first attempts to use formal techniques to solve problems specific to communication protocols, see Billington et.al (1988), Diaz (1982), Diaz (1987) or Merlin (1976). The main reasons for this were that: (i) Petri nets allow one to describe protocols

---

[1]Appeared in: *Protocol Specification, Testing and Verification, XV*. Eds. P.Dembiński, M.Średniawa, Eds., Chapman & Hall, 1996, pp.71–86.

1

in a very adequate way (in particular, by directly supporting the fundamental notions of concurrency and asynchrony which are inherent to communication protocols); (ii) there exists a rich body of models, verification techniques and computer-aided tools based on Petri nets; and (iii) the visually appealing graphical interface makes Petri nets easy to understand and manipulate for a wide range of practitioners. Having said that, in the past few years Petri nets lost much of their following in the protocol engineering community. We believe that mainly due to that fact that for a long time Petri nets did not fully support compositionality. This resulted in a failure to deal with large and complex designs like hierarchical protocol systems (or distributed systems) since nets corresponding to industrial-size protocols would often be too large to handle.

In the past few years, there has been a substantial progress in combining compositionality with Petri nets. In particular, a number of results have been obtained within the European BRA project Demon and its successor Caliban, see Best et.al (1993), Best and Hopkins (1993) and Koutny (1994). A similar, more protocol-oriented research has been undertaken in Anisimov (1991a,1991b) and Anisimov et.al (1993). We feel that the results obtained there allow us to make much more optimistic evaluation of the suitability of Petri nets for protocol engineering, and that one can apply Petri nets to the design of communication protocols in truly compositional style.

In this paper, after a short discussion in which we point out that compositionality is an inherent feature of protocols, and as such should be supported by adequate formal basis, we outline a systematic approach to the design of protocol systems. We here focus on the specification phase which is meant to be used as input for other phases (e.g. verification and testing). It is our intention to cover the latter in future, and some initial steps will be made in this paper (e.g. the definition of protocol correctness).
At the top level of design, we use entity calculus which comprises the notion of Petri net entity and a set of operations; in particular, the operation of concurrent composition. The external behaviour of entities is characterised using the notion of a bisimulation equivalence based on the step sequence semantics. At a lower level of design, we show how entities can be constructed from protocol procedures, using suitable composition rules, such as sequence and disabling. The correctness of composition operators is briefly addressed by discussing the relationship between the syntactic and behavioural notions of compositionality. The Appendix contains formal definitions.

Throughout the paper we illustrate the discussion using a toy protocol example from Merlin (1976), shown in Figure 1. Despite the fact that it is trivial, it enables us to illustrate the main ideas of the paper. In the nearest future, we plan to produce a specification of a simplified transport protocol of Bochmann (1989/90).
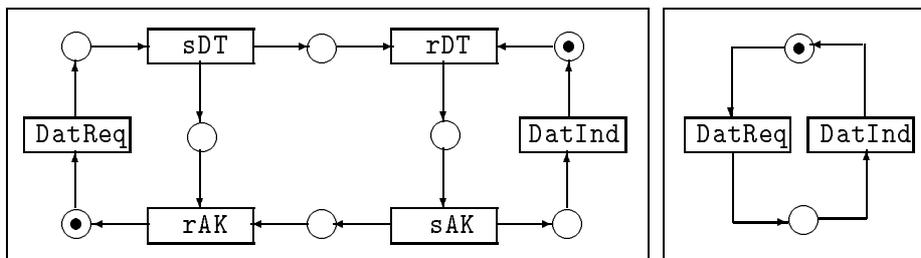

Figure 1: Toy protocol specification and its service (running example).

2 PROTOCOLS AND COMPOSITIONALITY

The standard protocols and protocol systems are inherently compositional. We can identify at least three features pertaining to their compositionality: (i) First of all, the system itself consists of a set of *entities*. Some correspond to different layers in the standard OSI model (ISO, 1983), e.g. the link, network, transport and session layers. Others are playing an auxiliary role, e.g. the control and timer entities. Entities are interconnected, through *service access points*, in accordance with the reference model. (ii) An entity usually has quite a complex internal structure which comprises a set of components – procedures. In particular, a protocol–entity will usually be built from several components, such as sub-protocols, phases, and protocol-procedures. (The term protocol-procedure will here be used to mean a part of the protocol which supports a single protocol function.) Typical examples of procedures would include connection request, data flow control and disconnection. There are special composition rules for combining different procedures, such as sequential composition (whereby one procedure can start to operate only after a successful termination of another procedure). Other examples of composition rules include iteration, parallel composition and disabling. (iii) Finally, each protocol-procedure will have its own internal structure which is constructed from *primitives* such as service primitives and protocol data units. The procedure specification defines all possible ways in which these primitives can be executed.

Thus we can distinguish three levels of compositionality in protocol design, which we will call *system*, *entity* and *procedure* levels. The problem of logical and functional correctness has to be addressed throughout the entire design process. This implies that each level should be supported by appropriate verification techniques that will guarantee the correctness of the resulting system. Both from the design and verification point of view, it is essential that the same formal basis be used at different levels. In this way, the results obtained at a lower level can be directly employed on the higher level(s) of the compositionality hierarchy.

3 ENTITIES AND THE SYSTEM LEVEL

A central notion used at the system level is that of an *entity*, see Anisimov (1989, 1991b) or Anisimov et.al (1993). The notion of a *protocol* entity occupies an important place in the OSI reference model, and is defined to be a logical module that operates performing certain protocol functions. Its execution involves communication with other entities, e.g. protocol entities of the neighbouring lower and higher levels. The communication is defined using *service access points*. As a result, an entity is a logical module which has explicitly indicated points of communication, called access points, through which it can communicate with other entities.

In the diagrams, entities will be represented as boxes with short adjacent lines, each line representing one access point. The composition of entities can then be represented by joining the corresponding lines. For example, Figure 2 shows a possible representation of our example toy protocol which is composed of three entities: sender (S), transfer media (M) and receiver (R).
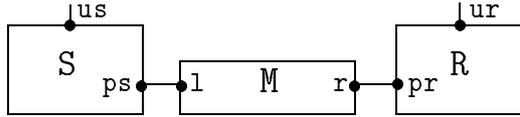
Figure 2: Architecture of example toy protocol.

We now will show how to represent entities and operations on them. We will represent entities as suitable Petri nets equipped with an additional information about the communication access points. All the formal definitions can be found in the Appendix.

3.1 Entities

An entity is a pair $E = \langle \Sigma, , \rangle$ where Σ is a Petri net and , is a finite set of labellings called access points. Each $\sigma \in$, is a mapping, $\sigma : T \rightarrow \mathcal{M}(\mathcal{A}_\sigma) \cup \{\tau\}$, where T is the transition set of Σ and $\mathcal{M}(\mathcal{A}_\sigma)$ is a set of non-empty multisets of elementary communication primitives (or names), $\mathcal{A}_\sigma$. σ associates with each transition a multiset of elementary primitives communicating through this access point, or a reserved symbol τ which represents transitions which are 'invisible' from the access point σ. In the diagrams, E will be represented by a Petri net whose each transition is labelled by a multiset of elementary names, each name being preceded by the name of an access point. The τ-labels will be omitted. For instance, if E has three access points α, β, γ and transition t labelled by $\alpha : a$, $\alpha : b$, $\alpha : b$ and $\beta : c$, then in the access point α transition t is labelled by multiset $\{a, b, b\}$ and in β by $\{c\}$. In the access point γ, t is labelled by τ (essentially, this means that executing t has no direct effect on any entity with which the communication is established using only γ). An entity E with access points $\alpha_1, \ldots, \alpha_n$ will be denoted as $E[\alpha_1, \ldots, \alpha_n]$.

Each access point contains information about the way in which an entity can communicate with another entity sharing this access point. Clearly, an entity can communicate differently using different access points. In particular, the same transition can be visible from one access point and invisible from another one. Our definition of entity labelling allows one to represent a simultaneous execution of *several* logical actions by firing only *one* transition. This is possible because in each access point the firing of a transition may correspond to the execution of a multiset of elementary communication actions. Moreover, the transition may define several such multisets, each of them corresponding to a distinct access point.

Figure 3, describes the entities used in Figure 2. Note that in Figure 3 we use explicitly multi-labelled transitions and, as a result, we will eventually obtain a slightly different (but equivalent) specification from that shown in Figure 1.

The sender is defined by entity S with two access points, us and ps, to connect respectively with its user and the transfer media. Its first transition, t_1, is 'visible' from both access points and corresponds to receiving service primitive DatReq from the user and sending protocol data unit sDT into media. Transition t_2 corresponds to receiving PDU rAK from the media (access point ps). From the access point us this transition is invisible and, consequently, its firing does not influence the user. The merging of two logical actions, DatReq and sDT, into a single 'physical' transition embodies a natural idea that there is no need to separate these two actions executed at two different access points.

S[us,ps]

us:DatReq
ps:sDT
t1

t2

ps:rAK

(i)

M[l,r]

l:sDT   r:rDT
t3   t5

t4   t6
l:rAK   r:sAK

(ii)

R[ur,pr]
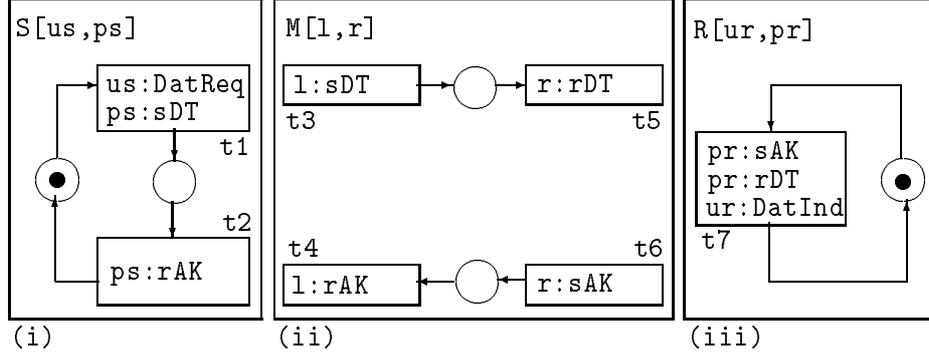
pr:sAK
pr:rDT
ur:DatInd
t7

(iii)

Figure 3: Entities used in the toy protocol: sender (i), media (ii) and receiver (iii).

The transfer media is represented by entity M with two access points, l and r, used respectively for the communication with its left and right entities. Transitions t_3 and t_4 are visible only from the left access point, and t_5 and t_6 only from the right access point (for brevity, we omitted the (marked) complement places of the two (unmarked) places which are shown, as the former become redundant once the three nets are composed together).

The receiver is represented by entity R with two access points, ur and pr, for the communication with its user and media. It contains only one transition, t_7. In access point pr, it is labelled with PDUs rDT and sAK which corresponds to simultaneously receiving data and sending acknowledgment through the access point pr. In the access point ur, transition t_7 is labelled by DatInd which corresponds to delivering the data to the user. Notice that instead of having three separate transitions, as it would have to be done if multiset labelling of transitions were not allowed, we have only one which is not merely a notational convenience, but also leads to the reduction of the state space of the resulting system, with clear benefits for the verification stage.

3.2 Operations on entities

The main operation for manipulating entities is that of concurrent composition. Let $E_1 = \langle \Sigma_1, , _1 \rangle$ and $E_2 = \langle \Sigma_2, , _2 \rangle$ be two entities with access points $\alpha \in , _1$ and $\beta \in , _2$. Then composing the two entities w.r.t. the access points α and β yields the entity:

$$E = (E_1 \, _\alpha|_\beta \, E_2) = \langle (\Sigma_1 \, _\alpha\|_\beta \, \Sigma_2), \{\hat{\gamma} \mid \gamma \in , _1 \cup , _2 \setminus \{\alpha, \beta\}\}\rangle$$

where $\Sigma_1 \, _\alpha\|_\beta \, \Sigma_2$ is a parallel composition of the two Petri nets such that the synchronisation between two sets of transitions, coming from Σ_1 and Σ_2, is carried out provided their visibility through respectively α and β is the same. The visibility of the new transitions w.r.t. the remaining access points (denoted by $\hat{\gamma}$) is defined as the multiset sum of the labellings of the constituent transitions (see the Appendix). The access points of the resulting entity is the union of the (suitably adjusted) original access points but without those which took part in the composition.

Figure 4 shows the result of performing the composition operation on the three entities from Figure 3 according to the linking given by Figure 2. As a result, we obtain the entity $TE[us, ur] = (S \, _{ps}|_l \, M \, _r|_{pr} \, R)$ with two access points for protocol users, us and ur. Note

that in applying the composition operation, transition t_1 was merged with t_3, t_2 with t_4, t_5 and t_6 with t_7.
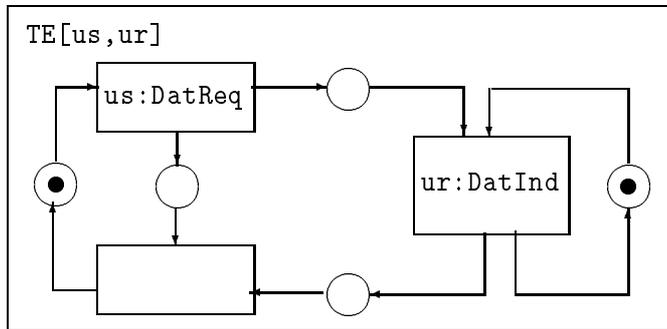

Figure 4: Composition of entities.

Sometimes an access points may happen to be redundant. In such a case, the operation of *abstraction* can be applied. Let $H \subseteq$, be a set of access points of an entity $E = \langle \Sigma, , \rangle$. Then the abstraction operation results in the new entity, $\tau_H(E) = \langle \Sigma, , \setminus H \rangle$. The operation does not change the underlying net and, consequently, its behavior at the remaining access points.

The possibility of representing entities and operations on them both in schematic and net-based forms results in a flexible protocol specification tool. At the higher level, the protocol architecture can be specified using schematic diagrams, as shown in Figure 2. At the lower level, each entity can be specified using a Petri net, as shown in Figure 3. The explicit representation of protocol architecture and the possibility to use different styles of specification (Vissers et.al, 1988) enhances the quality of specifications.

Note that in the specification carried out on the system level, the operation of composition is only indicated but not executed in the net-based form. Its actual execution, resulting in merging of the nets, may be needed for dealing with other tasks of protocol engineering such as analysis, verification and implementation. For instance, the analysis of the toy protocol against its logical correctness specification (e.g. absence of deadlocks) would involve deriving the entity $(S_{ps}|_l M_r|_{pr} R)$ and testing it for the presence of deadlocks or other incorrect situations. It is worth noting that the resulting net is more compact than the original one, depicted in Figure 1. The same can be said about the corresponding reachability graphs. Thus allowing multiset labelling can reduce the size of nets representing protocol systems.

3.3 Equivalence of entities

For solving the general problem of protocol verification, which usually amounts to demonstrating a correspondence between a protocol and its service, we need a notion of entity equivalence allowing us to compare the behaviour of different entities. The definition below is based on the notion of bisimulation (Milner, 1989).

Two entities E_1 and E_2 are bisimulation equivalent if there is a relation $\Re$ on the reachable markings of the underlying nets which is a (step) bisimulation relation w.r.t. transition labellings defined by pairs of corresponding access points (see the Appendix).

6

Such an equivalence is preserved by the entity composition and abstraction operations. As a result, one can substitute one entity in a system by another, equivalent one, without changing the behaviour of the whole system.
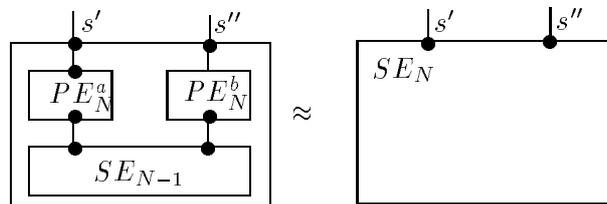

Figure 5: Protocol verification.

The verification of an N-layer protocol can consist then in comparing the behaviour of two entities – the N-service entity and its logical implementation. The latter is constructed as the composition of two protocol entities, PE_N^a and PE_N^b, with the entity of the (N-1)-service, SE_{N-1}, as shown in Figure 5. Assuming that we also have an entity which specifies a reference N-service, SE_N, the verification problem of the N-protocol reduces to proving that SE_N is bisimulation equivalent to the composition of PE_N^a, PE_N^b and SE_{N-1}. The N-protocol which is specified by entity PE_N is considered to be correct w.r.t. services SE_N and SE_{N-1} if such an equivalence is valid. For instance, the logical implementation of the toy protocol shown in Figure 4 is equivalent to its service shown in Figure 1.

One can also deal with the operation of hierarchical composition of entities in a very natural way. Indeed, if $PE_N[s_{N-1}, s_N]$ and $PE_{N+1}[s_N, s_{N+1}]$ are respectively protocol entities of the N- and (N+1)-layers, then the entity that would correspond to their hierarchical composition is built as $PE_{N,N+1} = (PE_N \ _{s_N}|_{s_N} \ PE_{N+1})$. Moreover one can show that:

> If the N-protocol defined as entity PE_N is correct w.r.t. services SE_N and SE_{N-1} and the (N+1)-protocol defined as entity PE_{N+1} is correct w.r.t. services SE_{N+1} and SE_N then the hierarchical protocol $PE_{N,N+1}$ is also correct w.r.t. services SE_{N+1} and SE_{N-1}.

Such a result formally justifies the layering structuring principle in OSI.

4 COMPOSITION RULES FOR THE ENTITY LEVEL

A protocol entity may itself have quite complex internal structure which will be described as a set of procedures combined together by means of suitable composition rules. A typical procedure will communicate with other entities in the same way as the enclosing entity and, therefore, will have the same set of access points. To allow joining procedures into entities, some composition rules are needed which require additional information about the states of the procedures. For instance, to perform sequential composition of two procedures, we need to know their initial and final states. Such an information will be formalised using the notion of a *macrostate*, which is a set of markings, see Anisimov (1991a) and Anisimov et.al (1993).

A macrostate of a net $N = \langle S, T, F \rangle$ is a set of 1-safe markings of N (in what follows, a 1-safe marking M is treated as the sets of places s for which $M(s) = 1$). We will say that a Petri net $\langle N, M \rangle$ is in a macrostate m if its current marking M belongs to m. A protocol procedure is then a tuple $R = \langle N, , , \Pi \rangle$ where $N = \langle S, T, F \rangle$ is a net, , is a set of access points and $\Pi = \langle h, l, r \rangle$ are macrostates called respectively the *head* (always a singleton set), *tail* and *reachable* states. Thus, a protocol procedure is defined like an entity without the initial marking, but instead equipped with three states carrying the information needed later for application of various composition operators.

In the diagrams, protocol procedures will be represented as entities with additional information specifying their states. The places marked at the only marking of a head state will indicated by short incoming arcs. If a tail state comprises only one marking then the places marked by it will be indicated by short outgoing arcs.

The sender and receiver entities of the toy protocol (see Figure 3) can be thought of as protocol procedures of a more complex multiphase protocol, containing additionally a connection establishment and disconnection procedures. These procedures, shown in Figure 6, have two access points: u for communicating with the user and p for communicating with the transfer media. Note that in the disconnect procedure, the actions of the incoming PDU disconnect request rDR, indicating to the user DisInd and sending disconnect confirmation sDC, are specified simultaneously as one transition.

To obtain an entity from a set of protocol procedures we use five composition rules: sequential and parallel compositions, iteration, disabling and transformation into an entity. All but the last one are based on a basic net operation of macrostate merging.

If N has macrostates v and w then merging v and w yields a net $\oplus_w^v(N)$, such that the places corresponding the two macrostates are 'multiplied' and the transitions adjacent to them 'split' (see the Appendix). Since the operation changes the set of places, any existing macrostate u of N has to be adjusted to yield a new macrostate $\oplus_w^v(u)$. Intuitively, after merging v and w, upon reaching any marking in one of the macrostates, the net may continue as though it were restarted in any of the markings of the other macrostate.

Let $P_1 = \langle N_1, , _1, \langle h_1, l_1, r_1 \rangle \rangle$ and $P_2 = \langle N_2, , _2, \langle h_2, l_2, r_2 \rangle \rangle$ be protocol procedures. Then the composition operations are defined as follows (c.f. Figure 7):

- Sequential composition: $(P_1; P_2)$. The tail state of P_1 is merged with the head state of P_2. The head state of the new net is h_1; the tail state is l_2.

- Parallel composition: $(P_1 ||| P_2)$. The nets of P_1 and P_2 are simply juxtaposed. The states are formed as all possible combinations of markings of the corresponding states of the two procedures.

- Iteration: $*(P_1)$. Iteration simply merges the head and tail states.

- Disabling: $(P_1 [> P_2)$. The reachable state of P_1 is merged with the head state of P_2. Thus the second procedure can start its execution at any time.

- Transforming into an entity: $\mathbf{E}(P_1) = \langle \langle N_1, M \rangle, , _1 \rangle$ where $\{M\} = h_1$.
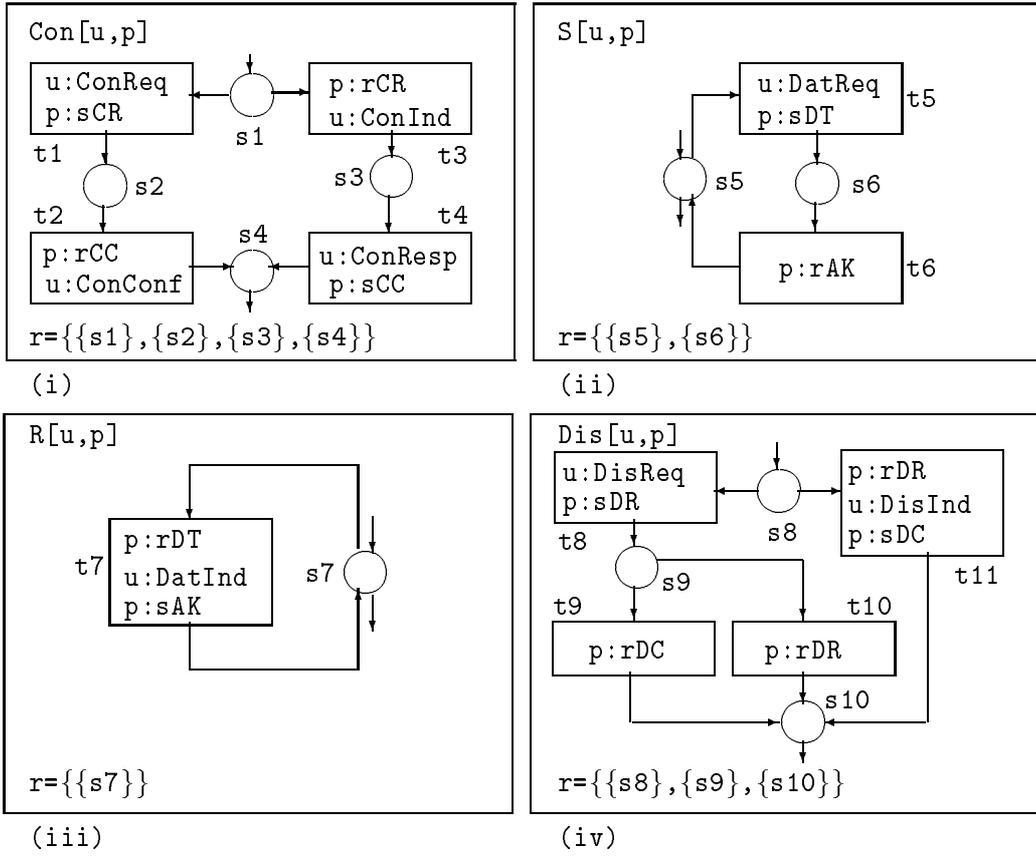
Figure 6: Protocol procedures: connection establishment (i), sending data (ii), receiving data (iii) and disconnecting (iv).
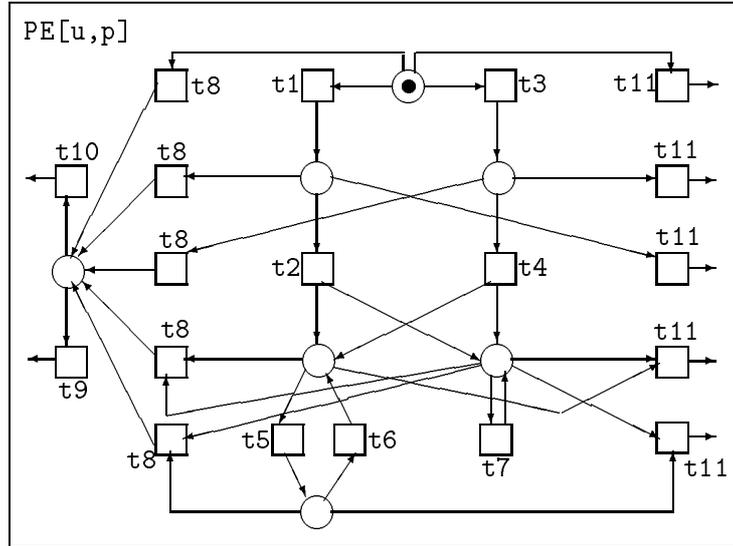

Figure 7: Composition of protocol procedures $\mathbf{E}(*((Con;(S|||R))[> Dis))$ from Figure 6 (arrows with no target node end at the only marked place).

9

5 COMPOSITIONALITY OF BEHAVIOURS

A successful approach to protocol engineering should provide a formal support for verification at every stage of design. It has long been recognised that dealing with a complex, highly concurrent computing system, such as a protocol system, makes the verification problem very hard since its state space is in general too large to handle. It is worth pointing out in this context that labelling transitions with 'simultaneous' actions can reduce the size of the formal representation of a system, and consequently ease the verification effort.

The basic idea behind using compositionality in behaviour verification is that of taking advantage of the structural properties of the system in order to decompose the verification process into smaller, more manageable pieces. For this to work, the operations used should enjoy desirable behavioural properties. For example, suppose that an entity E is a sequential composition of two procedures, A and B. Then one would expect that every terminating behaviour of E is of the form $\beta_A; \beta_B$, where β_A is a terminating behaviour of A and β_B is a terminating behaviour of B and ; is a concatenation operation for behaviours. In essence, what one should require of the operators used in compositional setting is that the behaviour of the resulting system should be a meaningful composition of the behaviours of the constituent subsystems.

The above, highly desirable, property has been investigated in Koutny (1994,1995) and Koutny et.al (1994). It turns out that one can formulate simple conditions about net operators which ensure behaviour compositionality. More specifically, suppose that f is a net operator. Then it is possible to show that for any valid application $N = f(N_1, \ldots, N_k)$, where the N_i's are marked and/or unmarked nets, the following holds:

1. If T_i is a step for N_i $(i \in I)$ then a combination of the T_i's is a step for N.

2. If T is a step for N then there are nets $N'_1, \ldots, N'_k$ such that $f(N_1, \ldots, N_k) = f(N'_1, \ldots, N'_k)$ and T is a combination of some steps T_i generated by the nets N'_i.

The above is rather strong property. To begin with, it allows one to describe the behaviour of $f(N_1, \ldots, N_k)$ as a simple composition of the behaviours of the nets $N_1, \ldots, N_k$. The result can be formulated for the interleaving, step sequence and causal partial order semantics, see Koutny (1994) and Koutny et.al (1994). Moreover, one can derive in an automatic way a process algebra, similar to CCS (Milner, 1989) or ACP (Baeten and Weijland, 1990), together with suitable inference rules and equivalences, which is operationally equivalent (w.r.t. bisimulation equivalence) to the compositionally defined nets. An important consequence of this is that there is a close relationship between two models of concurrent behaviour, Petri nets and process algebras. In particular, this opens a way for different tools and verification techniques developed for process algebras to be imported into the Petri net based framework.

6 CONCLUDING REMARKS

There are other issues in protocol engineering which we believe could be successfully addressed by extending the framework presented in this paper. The first is to extend the current formalism to include manipulation on data and dynamic configuration modelling. We expect that both problems, which are orthogonal to the main subject of this paper, can be dealt with using a suitable high level Petri net model. Another issue is that of a linguistic specification environment for our Petri net based model, similar to LOTOS (Bolognessi and Brinksma, 1987) or Estelle (Budkowski and Dembinski, 1987). Some general results in this area do already exist, e.g. Anisimov (1991b), Barbeau and Bochmann (1993) or Best and Hopkins (1993), but we feel that they need some modification in order to better fit the area of application. Finally, the problem of verification, i.e. checking the bisimulation equivalence, deserves due attention. We envisage that a number of existing verification techniques, such as those in Godefroid and Wolper (1993), Valmari (1990) and Vuong and Cowan (1982), could be improved by taking advantage of compositional definition of nets. Finally, a strong link with process algebras should allow one to import process algebra specific verification techniques (e.g. axiomatisations of behavioural equivalences) into the Petri net based framework.

7 ACKNOWLEDGEMENT

8 REFERENCES

Anisimov, N.A. (1989) *A Notion of Petri Net Entity for Communication Protocol Design.* Report, Institute for Automation and Control Processes, Russian Academy of Sciences, Vladivostok.

Anisimov, N.A. (1991a) *An Algebra of Regular Macronets for Formal Specification of Communication Protocols.* Computers and Artificial Intelligence **10**, 541–560.

Anisimov, N.A. (1991b) *A Petri Net Entity as a Formal Model for LOTOS, a Specification Language for Distributed and Concurrent Systems.* In: Parallel Computing Technologies (ed. N.N.Mirenkov), World Scientific, 440–450.

Anisimov, N.A., Kovalenko, A.A. and Postupalski, P.A. (1993) *Two–Levels Formal Model for Protocol Specification Based on Petri Nets.* In: Network Information Processing Systems. Proc. of the IFIP TC6 Int. Symp. (ed. K.Boyanov), Bulgarian Academy of Sciences, 143–154.

Baeten, J.C.M. and Weijland, W.P. (1990) *Process Algebra.* Cambridge Tracts in Theoretical Computer Science.

Barbeau, M., Bochmann, G.V. (1993) *A Subset of Lotos with the Computational Power of Place/Transition–Nets.* In: Proc. of 14th Int. Conf. Application and Theory of

Petri Nets, (ed. M.Ajmone Marsan), Springer-Verlag Lecture Notes in Computer Science 691, Springer.

Best, E., Devillers, R. and Hall, J.G. (1992) *The Box Calculus: a New Causal Algebra with Multi–label Communication.* In: Advances in Petri Nets 1992, Springer-Verlag Lecture Notes in Computer Science 609, 21–69.

Best, E. and Hopkins, R.P. (1993) $B(PN)^2 - a$ *Basic Petri Net Programming Notation.* Proc. of PARLE-93, Springer-Verlag Lecture Notes in Computer Science 694, 379–390.

Billington, J., Wheeler, G.H. and Wilbur–Ham, H.C. (1988) *PROTEAN: A High–level Petri Net Tool for the Specification and Verification of Communication Protocols.* IEEE Trans. Soft. Eng. **14**, 301–315.

v.Bochmann, G. (1989/90) *Specification of a Simplified Transport Protocol Using Different Formal Description Techniques.* Computer Network and ISDN Systems **18**, 335–377.

Bolognessi, T. and Brinksma, E. (1987) *Introduction to the (ISO, 1993) Specification Language LOTOS.* Computer Network and ISDN Systems **14**, 25–29.

Budkowski, S. and Dembinski, P. (1987) *An Introduction to Estelle: A Specification Language for Distributed Systems.* Computer Network and ISDN Systems **14**, 3–23.

Diaz, M. (1982) *Modeling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models.* Computer Networks **6**, 419–441.

Diaz, M. (1987) *Petri Net Based Models in the Specification and Verification of Protocols.* Springer-Verlag Lecture Notes in Computer Science 255, 135–170.

Godefroid, P. and Wolper, P. (1993) *Partial-order Methods for Temporal Verification.* In: Proc. of Proc. Concur'93, Springer-Verlag Lecture Notes in Computer Science 715, 233–246.

ISO (1983), IS 7498, Information Processing Systems – Open System Interconnections – Basic reference Model.

Kotov, V.E. (1978) *An Algebra for Parallelism Based on Petri Nets.* Springer-Verlag Lecture Notes in Computer Science 64, 39–55.

Koutny, M. (1994) *Partial Order Semantics of Box Expressions.* In: Proc. of 15th International Conference on Application and Theory of Petri Nets, Springer-Verlag Lecture Notes in Computer Science 815, 318-337.

Koutny, M., Best, E. and Esparza, J. (1994) *Operational Semantics for the Petri Box Calculus.* In: Proc. of 4th International Conference on Concurrency Theory, Springer-Verlag Lecture Notes in Computer Science 836, 210-225.

Koutny, M. (1995) *Compositional Definitions of Petri Nets.* A manuscript.

Merlin, P.M. (1976) *A Methodology for The Design and Implementation of Communic–ation Protocols.* IEEE Trans. Commun. **24**, 614–621.

Milner, R. (1989) *Communication and Concurrency.* Prentice Hall.

Olderog, E.R. (1987) *Operational Petri Net Semantics for CCSP.* In: Advances in Petri Nets 1987 (ed. G. Rozenberg), Springer-Verlag Lecture Notes in Computer Science 266, 196-223.

Rudin, H. (1987) *Network Protocols and Tools to Help Produce Them.* In: Ann. Rev. Comput. Sci. **2**, 291–316.

Valmari, A. (1990) *A Stubborn Attack on State Explosion.* Computer-Aided Verification, AMS-ACM DIMACS Series in Discrete Mathematics and Theoretical Computer Science **3**, American Mathematical Society, 25-41.

Vissers, C.A., Scollo, G. and Van Sinderen, M. (1988) *Architecture and Specification Style in Formal Description of Distributed Systems.* In: Proc. of PSTV VIII (eds. S.Aggarval and K.Sabnani), North–Holland.

Vuong, S.T. and Cowan, D.D. (1982) *A Decomposition Method for the Validation of Structured Protocols.* In: Proc. of INFOCOM'82, 209–220.

9 APPENDIX

A *multiset* over a set $\mathcal{A}$ is a mapping $\mu \colon \mathcal{A} \to \mathbf{N}$. $\mathcal{M}(\mathcal{A})$ will denote the set of non-empty finite multisets over $\mathcal{A}$. The sum of multisets $\mu_1, \ldots, \mu_k$ is defined as $(\mu_1 + \cdots + \mu_k)(a) = \mu_1(a) + \cdots + \mu_k(a)$, for all $a \in \mathcal{A}$. If $A = \{\mu_1, \ldots, \mu_k\}$, then $\Sigma_{\mu \in A} = \mu_1 + \cdots + \mu_k$.

A *Petri net* is a triple $N = \langle S, T, F \rangle$, where S and T are disjoint finite sets of respectively places and transitions and F is a flow relation, $F \subseteq (S \times T) \cup (T \times S)$. It is assumed that $^\bullet t$ and $t^\bullet$ are non-empty sets, for all $t \in T$, where $^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$. The dot-notation is extended in the usual way to sets of transitions and places. Ind_Σ will denote the set of all non-empty sets of mutually independent transitions (transitions t and u are independent if $(^\bullet t \cup t^\bullet) \cap (^\bullet u \cup u^\bullet) = \emptyset$). A *marked net* is a pair $\Sigma = \langle N, M \rangle$ where $M : S \to \mathbf{N}$ is a marking.

We adopt the standard step sequence semantics of a marked net $\Sigma = \langle S, T, F, M \rangle$. A finite non-empty set of transitions (step) U is enabled at marking M, if for all $s \in S$, $M(s) \geq |s^\bullet \cap U|$. It can be executed leading to a new marking M' defined, for every $s \in S$, by $M'(s) = M(s) - |s^\bullet \cap U| + |^\bullet s \cap U|$. We denote this by $M[U\rangle M'$. A step sequence is $\omega = U_1 \ldots U_k$ ($k \geq 0$) such that $M[U_1\rangle M_1 \ldots [U_k\rangle M_k$ for some $M_1, \ldots, M_k$. We will call M_k *reachable* from M, $M_k \in [M\rangle$. If $M(S) \subseteq \{0, 1\}$ then M is a 1-safe marking which will be identified with the set of the places for which its value is 1. The net is *1-safe* if all markings $[M\rangle$ are 1-safe.

9.1 Petri net entities

A Petri net *entity* is a pair $E = \langle \Sigma, , \rangle$, where $\Sigma = \langle S, T, F, M \rangle$ is a 1-safe marked net, and , is a finite set of *access points*. Each access point $\sigma \in ,$ is a mapping $\sigma \colon T \to \mathcal{M}(\mathcal{A}_\sigma) \cup \{\tau\}$, where $\mathcal{A}_\sigma$ is a set of elementary names of communication primitives, and τ is a reserved symbol. We will assume that distinct entities are *disjoint*, which means that they have disjoint nets and disjoint sets of access points. Below $E_i = \langle \Sigma_i, , _i \rangle$ ($i = 1, 2$) are two entities such that $\Sigma_i = \langle S_i, T_i, F_i, M_i^0 \rangle$ ($i = 1, 2$).

Concurrent composition
Let $\alpha \in , _1$ and $\beta \in , _2$. The composition of E_1 and E_2 w.r.t. the access points α and β is defined as:
$$E_1 \ _\alpha|_\beta \ E_2 = \langle (\Sigma_1 \ _\alpha\|_\beta \ \Sigma_2), \{\widehat{\gamma} \mid \gamma \in , _1 \cup , _2 \setminus \{\alpha, \beta\}\} \rangle$$
In the above, $\Sigma_1 \ _\alpha\|_\beta \ \Sigma_2$ is the marked net $\langle S_1 \cup S_2, T, F, M_1^0 \cup M_2^0 \rangle$ whose transitions are

$T = \{t \in T_1 \mid \alpha(t) = \tau\} \cup \{t \in T_2 \mid \beta(t) = \tau\} \cup T^0$, where:

$$T^0 = \{t_{R_1 \cup R_2} \mid R_1 \in Ind_{\Sigma_1} \wedge R_2 \in Ind_{\Sigma_2} \wedge (\Sigma_{t \in R_1} \alpha(t)) = (\Sigma_{t \in R_2} \beta(t))\}$$

The flow relation is unchanged for the transitions not in T^0. For $t_{R_1 \cup R_2} \in T^0$, we have $(t_{R_1 \cup R_2}, s) \in F$ if $s \in S_1$ and $(R_1 \times \{s\}) \cap F_1 \neq \emptyset$, or $s \in S_2$ and $(R_2 \times \{s\}) \cap F_2 \neq \emptyset$. $(s, t_{R_1 \cup R_2})$ is defined in a similar way.

If $\gamma \in , _1 \setminus \{\alpha\}$ then we have (for $\gamma \in , _2 \setminus \{\beta\}$ the definition is similar):

$$\widehat{\gamma}(t) = \begin{cases} \gamma(t) & \text{if} \quad t \in T \cap T_1 \\ \tau & \text{if} \quad t \in T \cap T_2 \\ \Sigma_{u \in R_1} \gamma(u) & \text{if} \quad t = t_{R_1 \cup R_2} \in T^0 \end{cases}$$

Concurrent composition is commutative and associative (up to net isomorphism).

Abstraction
Let $H \subseteq , _1$ be a set of access points. Then $\tau_H(E_1) = \langle \Sigma_1, , _1 \setminus H \rangle$.

9.2 Bisimulation equivalence for entities

To introduce bisimulation equivalence we need to define what it means to execute an entity $E = \langle \Sigma, , \rangle$ w.r.t. an external observation specified by an access point $\alpha \in , .$ We first extend the notion of an access point to sets of transitions: Let $U \subseteq T$ and $V = \{t \in U \mid \alpha(t) \neq \tau\}$. Then $\alpha(U) = \Sigma_{t \in V} \alpha(t)$ if $V \neq \emptyset$, and $\alpha(U) = \tau$ otherwise.

Suppose now that M is a marking reachable from the initial marking of Σ and U is a step of transitions enabled at M leading to marking M', $M[U\rangle M'$. Then we will denote $M \xrightarrow{\alpha:\alpha(U)} M'$. Moreover, we denote $M \xrightarrow{\alpha:A} M'$ if one of the following holds:

(1) If $A \neq \tau$ then $M(\xrightarrow{\alpha:\tau})^* \circ \xrightarrow{\alpha:A} \circ (\xrightarrow{\alpha:\tau})^* M'$.

(2) If $A = \tau$ then $M(\xrightarrow{\alpha:\tau})^* M'$.

Let $|, _1| = |, _2|$ and let $\overline{\alpha} = (\alpha_1, \ldots, \alpha_k)$ and $\overline{\beta} = (\beta_1, \ldots, \beta_k)$ be enumerations of respectively $, _1$ and $, _2$. Then E_1 and E_2 are bisimilar w.r.t. $\overline{\alpha}$ and $\overline{\beta}$, $E_1 \,_{\overline{\alpha}}\!\approx_{\overline{\beta}} E_2$, if there exists a relation $\Re \subseteq [M_1^0\rangle \times [M_2^0\rangle$ such that $(M_1^0, M_2^0) \in \Re$ and for all i ($1 \leq i \leq k$) and $(M_1, M_2) \in \Re$, the following hold:

(1) If $M_1 \xrightarrow{\alpha_i:A} M_1'$ then there is M_2' such that $M_2 \xrightarrow{\beta_i:A} M_2'$ and $(M_1', M_2') \in \Re$.

(2) If $M_2 \xrightarrow{\beta_i:A} M_2'$ then there is M_1' such that $M_1 \xrightarrow{\alpha_i:A} M_1'$ and $(M_1', M_2') \in \Re$.

The equivalence relation is insensitive to the particular order in which the access points are enumerated. It is also possible to show that if $E_1 \,_{\overline{\alpha}}\!\approx_{\overline{\beta}} E_2$ and $E_2 \,_{\overline{\beta}}\!\approx_{\overline{\gamma}} E_3$, then $E_2 \,_{\overline{\beta}}\!\approx_{\overline{\alpha}} E_1$ and $E_1 \,_{\overline{\alpha}}\!\approx_{\overline{\gamma}} E_3$. What is more important is that the bisimulation equivalence is a congruence with respect to the abstraction and concurrent composition operators. The latter means that if $E_1 \,_{\overline{\alpha}}\!\approx_{\overline{\beta}} E_2$ and $F_1 \,_{\overline{\gamma}}\!\approx_{\overline{\delta}} F_2$, where $\overline{\gamma} = (\gamma_1, \ldots, \gamma_l)$ and $\overline{\delta} = (\delta_1, \ldots, \delta_l)$, then

$$(E_1 \,_{\alpha_1}|_{\gamma_1} F_1) \,_{\overline{\kappa}}\!\approx_{\overline{\mu}} (E_2 \,_{\beta_1}|_{\delta_1} F_2)$$

where $\overline{\kappa} = (\alpha_2, \dots, \alpha_k, \gamma_2, \dots, \gamma_l)$ and $\overline{\mu} = (\beta_2, \dots, \beta_k, \delta_2, \dots, \delta_l)$. Such a result is essential for hierarchical verification of communication protocols. Note that it would not hold if we used bisimulation based on interleaving semantics.

9.3 Protocol procedures

A protocol *procedure* is a tuple $P = \langle N, , , \Pi \rangle$ where $N = \langle S, T, F \rangle$ is a Petri net, , is finite set of access points, and $\Pi = \langle h, l, r \rangle$ are macrostates (non-empty sets of 1-safe markings of N) called respectively the *head, tail* and *reachable* states. $h = \{M\}$ is a singleton set, and the marked net $\langle N, M \rangle$ satisfies $[M\rangle \subseteq r$ and, for all $M' \in l \cup [M\rangle$ and $M'' \in [M\rangle$, if $M' \subseteq M''$ then $M' = M''$.

The intuitive meaning of h and l is that they comprise the valid initial and final markings of P. The third macrostate, r comprises all markings reachable from the initial marking in h (although there may be markings in r which will never be reached).

Procedure transformation
This operation takes P and produces entity $\mathbf{E}(P) = \langle N, M, , \rangle$.

Parallel composition
It takes two protocol procedures, $P_i = \langle N_i, , , _i, \langle h_i, l_i, r_i \rangle \rangle$ $(i = 1, 2)$ and yields

$$P_1 \| | P_2 = \langle N, , , _1 \cup , _2, \langle h_1 \otimes h_2, l_1 \otimes l_2, r_1 \otimes r_2 \rangle \rangle$$

where N is the disjoint union of N_1 and N_2 and $v \otimes w = \{M \cup M' \mid M \in v \land M' \in w\}$.

Macrostate merging
Let v, w be two non-empty macrostates such that $(^\bullet v \cup v^\bullet) \cap (^\bullet w \cup w^\bullet) = \emptyset$, where for each macrostate u, $^\bullet u = \bigcup \{^\bullet M \mid M \in u\}$ and $u^\bullet = \bigcup \{M^\bullet \mid M \in u\}$. The merging of v and w yields a new net $\oplus_w^v(N) = \langle S', T', F' \rangle$, where (below v and w denote respectively $\bigcup \{M \mid M \in v\}$ and $\bigcup \{M \mid M \in w\}$):

$$
\begin{aligned}
S' &= \{s \in S \mid s \notin v \cup w\} \cup \{\phi_{L,r}^{M,p} \mid M \in v \land p \in M \land L \in w \land r \in l\} \\
T' &= \{t \in T \mid (^\bullet t \cup t^\bullet) \cap v = \emptyset = (^\bullet t \cup t^\bullet) \cap w\} \qquad\qquad\qquad\quad \cup \\
& \quad \{t_L^M \mid t \in T \land M \in v \land L \in w \land (^\bullet t \cup t^\bullet) \cap v \neq \emptyset \neq (^\bullet t \cup t^\bullet) \cap w\} \quad \cup \\
& \quad \{t^M \mid t \in T \land M \in v \land (^\bullet t \cup t^\bullet) \cap v \neq \emptyset = (^\bullet t \cup t^\bullet) \cap w\} \qquad \cup \\
& \quad \{t_L \mid t \in T \land M \in w \land (^\bullet t \cup t^\bullet) \cap v = \emptyset \neq (^\bullet t \cup t^\bullet) \cap w\}
\end{aligned}
$$

and the flow relation between the places and transitions from N is unchanged. Moreover,

(1) If $s = \phi_{L,r}^{M,p}$ then $(s, t_L^M) \in F'$ if $(p, t) \in F$ or $(r, t) \in F$; $(s, t^M) \in F'$ if $(p, t) \in F$; and $(s, t_L) \in F'$ if $(r, t) \in F$.

(2) If $s \in S' \cap S$ then $(s, t_L^M) \in F'$ if $(s, t) \in F$ or $(s, t) \in F$; $(s, t^M) \in F'$ if $(s, t) \in F$; and $(s, t_L) \in F'$ if $(s, t) \in F$.

The arrows from transitions to places are defined in a similar way.

In addition to defining macrostate merging, we define, for every 1-safe marking M' of Petri net N,

$$\oplus_w^v(M') = (M' \cap S') \cup \{\phi_{L,r}^{M,p} \mid \{p,r\} \cap M' \neq \emptyset\}$$

The last definition is extended to macrostates in the obvious way and for every protocol procedure $P = \langle N, , , \langle h, l, r \rangle \rangle$ we then define

$$\oplus_w^v(P) = \langle \oplus_w^v(N), \oplus_w^v(,), \langle \oplus_w^v(h), \oplus_w^v(l), \oplus_w^v(r) \rangle \rangle$$

where the label given by an access point to a new transition resulting from the splitting of $t \in T$ (i.e. one of the transitions of the form t_L^M, t^M and t_L) is the same as that of t.

Other operations on procedures

An intuitive meaning of a sequential composition of two procedures, P_1 and P_2, is that the result, $P_1; P_2$, should first behave like P_1 and after terminating in one of its final markings, it should continue as P_2 started in its initial marking. Such a description of the meaning of the sequential composition should be matched by an appropriate definition on the net level (c.f. Kotov (1978) and Olderog (1987)). A situation which can pose here problems is the possibility of 'going back' to P_1 after passing the control to P_2. We will prevent this by putting a restriction on the arcs outgoing from (incoming to) places in the tail (head) markings of P_1 (P_2). This should not be considered as a major restriction because there are simple equivalence preserving transformation rules for protocol procedures (such as one step unwinding) allowing one to bypass them in practice. The same remark applies to the remaining operators used to combine procedures: choice, iteration and disabling.

Let $P_i = \langle N_i, , , _i, \langle h_i, l_i, r_i \rangle \rangle$ $(i = 1, 2)$ be two procedures such that $l_1^\bullet = \emptyset$ or $^\bullet h_2 = \emptyset$. Then the sequential composition of P_1 and P_2 is defined as

$$P_1; P_2 = \oplus_{h_2}^{l_1}(\langle N_1 \cup N_2, , , _1 \cup , _2, \langle h_1, l_2, r_1 \cup r_2 \rangle \rangle)$$

The choice operation is defined in a similar way. Iteration is defined as

$$*(P_1) = \oplus_{h_1}^{l_1}(\langle N_1, , , _1, \langle h_1, l_1, r_1 \rangle \rangle)$$

provided that $|l_1| = 1$, $l_1^\bullet = \emptyset = ^\bullet h_1$ and $|M| = 1$ for $M \in h_1 \cup l_1$. Finally, for disabling of P_1 by P_2 we assume that $^\bullet h_2 = \emptyset$ and then define:

$$P_1[> P_2 = \oplus_{h_2}^{r_1}(\langle N_1 \cup N_2, , , _1 \cup , _2, \langle h_1, l_1 \cup l_2, r_1 \cup r_2 \rangle \rangle)$$