Compositional Petri Nets in Protocol Engineering

Nikolay A. Anisimov and Maciej Koutny

N.Anisimov is with the Institute for Automation and Control Processes, Far East Branch of the Russian Academy of Sciences, 5 Radio Str., Vladivostok, 690041 Russia, E-mail: anisimov@iapu2.marine.su. Maciej Koutny is with the Department of Computing Science, University of Newcastle, Newcastle upon Tyne NE1 7RU, U.K., E-mail: Maciej.Koutny@newcastle.ac.uk.

Abstract

This paper addresses the problem of designing communication protocols within a framework based on Petri nets and supporting compositionality of structure and behaviour. Although Petri nets have for several years been applied as a formal model in which one could describe and analyse communication protocols, it can be argued that at present they are not widely used in protocol engineering. The main reason for this can be attributed to a lack of compositionality which meant that Petri nets were unable to deal with large computing systems. However, recent results on combining Petri nets and process algebras have radically changed that, and we here argue that Petri nets can provide an effective formal model for protocol engineering. After pointing 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. The top level of the design hierarchy is based on the notion of a Petri net entity and the operation of concurrent composition. The external behaviour of entities is characterised using the notion of a bisimulation equivalence. At the lower level of design, we show how entities can be constructed from protocol procedures using suitable composition rules, such as sequence, iteration and disabling. We then discuss the relationship between syntactical and semantical (behavioural) notions of compositionality.

Keywords

Communication protocols, protocol engineering, Petri nets, compositionality, synchronisation, protocol hierarchy, bisimulation equivalence, step sequence semantics

I. Introduction

It is generally recognised that the design of communication protocols is one of the most complex tasks in computer network development. The design methodology typically comprises various stages including specification, verification, performance analysis, implementation and testing. All these activities have led to the emergence of a new discipline, usually referred to as the *protocol engineering* [49], [61], [65], [77]. In its early days, most of the development was done following an intuitive approach and relied on the practical experience of the developers. However, it was not long before it became apparent that this was insufficient to cope with the inherent complexity of nontrivial communication protocols. To manage this complexity, protocol engineering turned to formal techniques which since have been applied to virtually all stages of protocol design. There were several attempts of applying different formalisms to deal with communication protocols, such as finite state automata, programming languages, process algebras and temporal logics [26], [66]. In this paper, we will concentrate the discussion on the theory of Petri nets, and will argue that it can provide a sound and effective formal basis for protocol engineering.

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 protocol engineering, see [7], [14], [31], [33], [34], [35], [50], [51], [52], [60]. The early results were promising and generated a feeling that Petri nets were going to solve most of the crucial problems in those areas of protocol engineering which are amenable to formal treatment. The main reason for this optimism was that:

- Petri nets allow one to describe protocols in an adequate and, at the same time, strictly mathematical way. In particular, Petri nets directly support the fundamental notions of concurrency and asynchrony which are inherent to communication protocols.
- There exists a rich body of models, verification techniques and computer-aided tools based on Petri nets [25].
- The visually appealing graphical representation makes Petri nets easy to understand and manipulate for a wide range of practitioners, even those with a limited background in formal methods.

Moreover, several Petri net models were introduced to allow one to describe manipulations on data, variables and parameters which is essential for the specification of many real-life protocols. Notable examples of such models are Evaluation nets [32], [56], Numerical Petri nets [8], [20], [21], [28], [69], Predicate/Transition nets [13], [27], [38], [74] and Coloured nets [43], [76].

Despite what we consider to be significant advantages of using Petri nets, they have lost in recent years much of their following within the protocol engineering community in favour of other formal description techniques such as SDL [64], Estelle [26], [36], LOTOS [23], [37] and PROMELA [41]. The latter are based on extended finite state automata and process algebras; typical examples of work employing these formalisms are [48], [49], [65]. We believe that the main reason for this shift was that for a long time Petri nets suffered from a lack of proper support of compositionality and modularity. This resulted in a failure to deal with large and complex designs like hierarchical protocol systems since

Petri nets corresponding to industrial-size protocols would often be too large to handle efficiently. (As the matter of fact, such a problem is not restricted to the domain of protocols but is also present in the applications of Petri nets to the design of other large scale systems, e.g., distributed systems and flexible manufacturing systems.) Some earlier attempts to bring compositionality and modularity into Petri nets [11], [12], [45], [54], [67], [71] did not result in a sufficient improvement. However, 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 [16], [17], [19], [46]. A similar, more protocol-oriented research has been undertaken in [2], [3], [5]. In this paper we take advantage of these recent developments in order to demonstrate that Petri nets can be successfully used in the specification and verification of communication protocols, and hierarchical protocol systems. We present a general compositional Petri net framework for dealing with protocols based on hierarchical approach. In particular, we introduce construction rules for protocols and formalise the notion of protocol correctness. We also outline how such a framework can support modular verification techniques, however their detailed discussion lies outside the scope of this paper.

The structure of the paper is as follows. In section II, after pointing out that compositionality is an inherent feature of communication protocols and as such should be supported by an adequate formal basis, we outline a systematic approach to the design of protocol systems. In section III, we present some basic notions related to Petri nets. Section IV introduces the top level of the proposed design hierarchy which is based on the notion of a Petri net entity and the operation of concurrent composition. The external behaviour of protocol entities is characterised using the notion of a bisimulation equivalence based on the step sequence semantics. In section V, a lower level of design is discussed. It is shown how protocol entities can be constructed from protocol procedures (a class of Petri nets) using suitable composition rules, such as sequential composition, iteration and disabling. Section VI is devoted to the procedure level of the design hierarchy. In section VII, we discuss the relationship between the syntactical and semantical (behavioural) notions of compositionality. We also comment on the existing link between compositionally defined Petri nets and process algebras. Section VIII contains a brief description of a computer-aided tool based on the approach presented in this paper.

The paper is a revised and extended version of the conference paper [6].

II. Protocols and Compositionality

The standard protocols and protocol systems are inherently compositional. Figure 1 illustrates a typical structure of a protocol system where we can identify the following features pertaining to compositionality:

- The system itself consists of a set of *entities*. Some correspond to different layers in the standard OSI model, [42], e.g., the link, network, transport and session layers. Other play an auxiliary role, e.g., the control and timer entities. The entities are interconnected, through *service access points*, in accordance with the reference model.
- An entity usually has a complex internal structure which comprises a set of components, called *procedures*. In particular, a protocol–entity will often be built from several components, such as sub-protocols, phases, and protocol-procedures. (The term protocol-procedure will be used to mean a part of the protocol which supports a single protocol function.) Typical examples of procedures are connection request, data flow control and disconnection. There are special rules for combining different procedures, such as sequential composition which ensures that one procedure can start to operate only after a successful termination of another procedure. Other examples of composition rules are iteration, parallel composition and disabling.
- Each protocol-procedure has its own internal structure which is constructed from *primitives* such as service primitives and protocol data units. The specification of a procedure 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:

- *System level.* The overall logical structure of the system is designed, using entities as the elementary building blocks.
- *Entity level.* Each entity is designed using procedures which are combined together using composition rules.

- *Procedure level.* Procedures are designed using a set of service primitives and protocol data units.

The problem of logical and functional correctness has to be addressed throughout the entire design process. This means that each level should be supported by appropriate verification techniques which would 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 directly be used at the higher level(s) of the compositionality hierarchy.

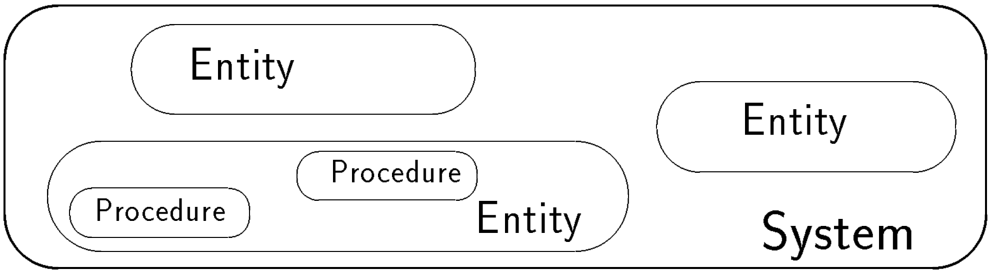

Fig. 1. Structure of a protocol system.

III. Petri nets

This section contains some basic definitions and notations concerning Petri nets. There are several easily accessible references covering these in more detail, e.g., [15], [55], [59], [62]. We also present a Petri net model of a simple communication protocol.

A. Basic Petri nets

A Petri net is a triple $N = \langle S, T, F \rangle$, where S and T are disjoint 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 disjoint sets, for all $t \in T$, where ${}^\bullet x = \{y \mid (y, x) \in F\}$ is the pre-set of $x \in S \cup T$ and $x^\bullet = \{y \mid (x, y) \in F\}$ is its post-set. The dot-notation is extended in the usual way to sets of transitions and/or places, e.g., ${}^\bullet X = \bigcup\{{}^\bullet x \mid x \in X\}$. All the nets considered in this paper have finite S and T. We also define a family of sets of independent transitions, Ind_Σ, which comprises all non-empty sets $U \subseteq T$ such that for every pair of distinct transitions t and u in U, $({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \emptyset$.

A *marked* net is a pair $\Sigma = \langle N, M_0 \rangle$ such that $N = \langle S, T, F \rangle$ is a Petri net and $M_0 \colon S \to \{0, 1, 2, \ldots\}$ is the initial marking. In general, a marking is a mapping $M \colon S \to \{0, 1, 2, \ldots\}$.

We use the standard rules about representing nets as directed graphs, viz. places are represented as circles, transitions as rectangles, the flow relation is indicated by directed arcs, and markings are shown by placing tokens inside circles.

B. Step sequence semantics

We will use the standard step sequence semantics of a marked net $\Sigma = \langle N, M_0 \rangle$. A finite non-empty set of transitions (step) U is enabled at a marking M of Σ, if for all $s \in S$, $M(s) \geq |s^\bullet \cap U|$. An enabled step U 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 finite step sequence for a marking M is $\omega = U_1 \ldots U_k$ $(k \geq 0)$ such that $M\ [U_1\rangle\ M_1 \ldots M_{k-1}\ [U_k\rangle\ M_k$ for some markings $M_1, \ldots, M_k$. We denote this by $M\ [\omega\rangle\ M_k$, and call the marking M_k reachable from M. The set of all marking reachable from M will be denoted by $[M\rangle$.

If $M(S) \subseteq \{0, 1\}$ then we will call M a 1-safe marking and identify it with the set of all the places for which the value of M is equal to 1. The marked net Σ is 1-safe if every marking reachable from its initial marking is 1-safe.

C. An example

Throughout the paper we illustrate the discussion using a toy protocol specified in Figure 2 that was used in [50] to demonstrate how Petri nets could be used to model protocols. Figure 2(i) shows a Petri net model of the protocol where the subnet comprising nodes s_1, s_2, s_3, t_1, t_2, t_3 corresponds to the sender part, the subnet comprising s_6, s_7, s_8, t_4, t_5, t_6 corresponds to the receiver, and the places s_4 and s_5 correspond to the transfer media. The protocol works as follows. On receiving a data block by executing a service primitive (SP) DatReq (transition t_1) from the upper level, the sender sends a protocol data unit (PDU) sDT (t_2) to the transfer media (place s_4) and enters a waiting state (s_3). On receiving PDU rDT (t_4), the receiver sends an acknowledgment, PDU sAK (t_5), delivers the received data to its user using an SP DatInd (t_6) and returns to its initial state (s_7).

On receiving the acknowledgment, PDU rAK (t_3), the sender returns to its initial state (s_1). Note that the last two transitions can be executed concurrently.

Figure 2(ii) shows the service that the above protocol is supposed to provide. The service is very simple and specifies that receiving data through SP DatReq (t_7) must be followed by its sending through SP DatInd (t_8). Note that the Petri net in (ii) is the specification of the protocol, which only takes into account two actions, DatReq and DatInd, and ignoring the remaining ones (more precisely, by treating them as internal or invisible). A formal device which allows one to make a distinction in the status between different transitions is *labelling*. We will later take a full advantage of this technique when access points of protocol entities and procedures will be defined. Referring to the diagram, the Petri net representing the service can be formally represented as

$$N = \langle \{s_9, s_{10}\}, \{t_7, t_8\}, \{(s_9, t_7), (t_7, s_{10}), (s_{10}, t_8), (t_8, s_9)\}\rangle.$$

The depicted marking can be represented either as the mapping M such that $M(s_9) = 1$ and $M(s_{10}) = 0$, or as the singleton set of places, $M = \{s_9\}$. For the net representing the protocol, we have ${}^\bullet t_2 = \{s_2\}$ and $t_2{}^\bullet = \{s_3, s_4\}$. The step sequence generated by this net in the above scenario is $\omega = \{t_1\}\{t_2\}\{t_4\}\{t_5\}\{t_3, t_6\}$.

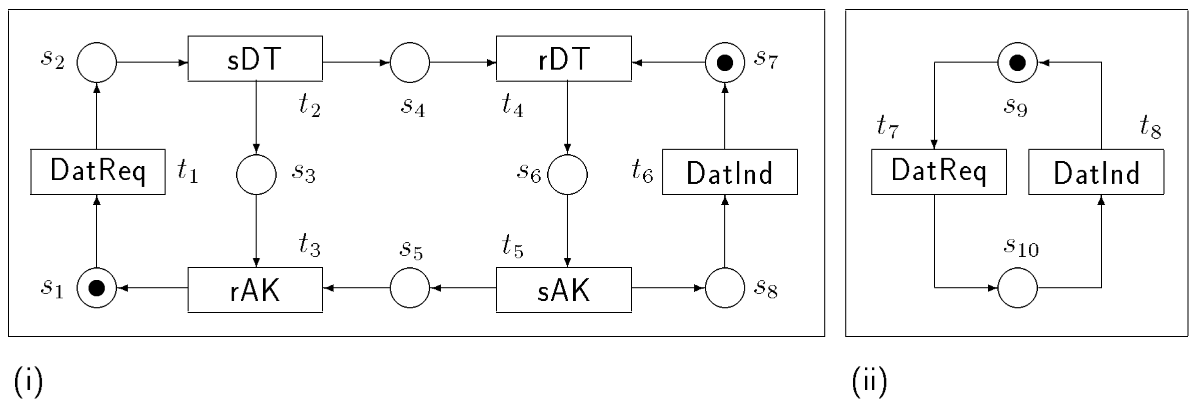

Fig. 2. Toy protocol specification (i) and its service (ii).

As we will see later, the example specification in Figure 2(i) does not satisfy the service in Figure 2(ii). Essentially, the specification behaves as a two-place buffer, whereas the service behaves as a one-place buffer. Of course, we could have changed the service net to reflect this, but later we will derive another specification based on the idea of the toy protocol, by following the approach presented in this paper, which will be more compact than that in Figure 2(i) and at the same time correct with respect to the service in Figure 2(ii).

IV. Entities and System level

In this section we discuss some of the formal notions which are needed to support compositionality at the system level [1], [3], [5]. As mentioned in Section II, the central notion used at this level is that of an entity. In the OSI reference model [42], an entity is defined as 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. I.e., an entity is a logical module with explicitly indicated points of communication, called access points, through which it can communicate with other entities.

In the schematic diagrams, entities will be represented as boxes with short adjacent edges, each edge representing one access point. The connectivity among entities can then be represented by joining the edges representing the corresponding access points. For example, Figure 3 shows a possible representation of the example protocol which is composed of three entities: sender (S), transfer media (M) and receiver (R). In the rest of this section, we will show how to model entities and operations on them in a formal way.

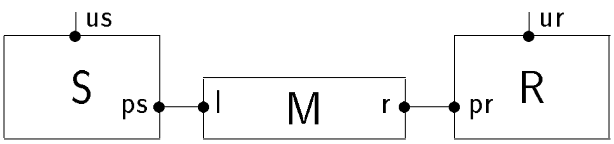

Fig. 3. Architecture of the example protocol.

A. Entities

We will represent an entity as a Petri net equipped with an additional information about the communication access points.

A Petri net entity (or entity) is a pair $\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$, where $\Sigma = \langle S, T, F, M_0 \rangle$ is a

1-safe marked net and acc is a finite set of labellings called access points. Each access point $\mathsf{a} \in \mathsf{acc}$ is a mapping $\mathsf{a}: T \to \mathsf{mult}(\mathsf{A})$, where A is a fixed set of elementary names of communication primitives and $\mathsf{mult}(\mathsf{A})$ is the set of all finite multisets over A (see the Appendix for the notations concerning multisets). Note that the empty multiset $\emptyset_{\mathsf{m}} \in \mathsf{mult}(\mathsf{A})$ represents an internal (or silent) action which is denoted by τ in CCS [53]. That is, a associates with each transition some elementary primitives communicating through this access point, or the empty multiset $\emptyset_{\mathsf{m}}$ which means that a transition is 'invisible'. We will assume that distinct entities have disjoint underlying nets.[1] Note that this implies that they have disjoint sets of access points. To be able to specify which access points should be connected together, we associate with each access point a its sort, denoted by Id_{a}. We extend the notion of a sort to sets of access points and entities, by respectively defining $\mathsf{Id}_{\mathsf{acc}} = \{\mathsf{Id}_{\mathsf{a}} \mid \mathsf{a} \in \mathsf{acc}\}$ and $\mathsf{Id}_{\mathsf{pne}} = \mathsf{Id}_{\mathsf{acc}}$. It is assumed that the sorts of distinct access points of an entity are different. Hence we can define, for every $\mathsf{id} \in \mathsf{Id}_{\mathsf{pne}}$, the access point $\mathsf{pne}_{\mathsf{id}}$ as being the unique $\mathsf{a} \in \mathsf{acc}$ such that $\mathsf{Id}_{\mathsf{a}} = \mathsf{id}$.

The intuition behind the sort of an access point is similar to that of a channel in CSP [40]; that is, it identifies a distinctive part of the communication interface of a system. In the schematic diagrams representing entities, it is the access points with the same sort which are joined by edges.

In the diagrams, an entity will be represented by a Petri net whose each transition is labelled by a multiset of elementary names, each elementary name being preceded by the name of an access point. For instance, if an entity has three access points, a, b and c, and a transition t is labelled by $\mathsf{a} : a$, $\mathsf{a} : b$ and $\mathsf{b} : c$ then $\mathsf{a}(t) = \{a, b\}$, $\mathsf{b}(t) = \{c\}$ and $\mathsf{c}(t) = \emptyset_{\mathsf{m}}$.[2] An entity pne with access points $\mathsf{a}_1, \ldots, \mathsf{a}_n$ will sometimes be denoted by $\mathsf{pne}[\mathsf{a}_1, \ldots, \mathsf{a}_n]$.

Each access point contains information about the way in which an entity can communicate with another entity having an access point of the same sort. Clearly, the entity can communicate in different ways using different access points. In particular, the same transition can be visible from one access point and invisible from another one. The way in which transitions are labelled allows one to represent a simultaneous execution of *several*

[1] This condition is not restrictive since Petri nets can be treated as defined up to isomorphism.

[2] Essentially, $\mathsf{c}(t) = \emptyset_{\mathsf{m}}$ means that executing t has no direct effect on any entity with which the communication is established using only access point c.

logical actions by executing only *one* transition. This follows from the fact that in each access point the execution 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 different access point. As it will be explained later, such a property can lead to a simplification (and reduction in size) of the description of a communication protocol systems.

In Figure 4, we give a Petri net description of the entities used in the protocol model of Figure 3. It is important to stress that in Figure 4 we use explicitly multi-labelled transitions and, as a result, we will eventually obtain a slightly different (actually, more compact and thus better) specification from that shown in Figure 2.

The sender is defined by entity S (see Figure 4(i)) with two access points, us and ps, to connect respectively with its user and the transfer media. It contains two transitions: The first one, 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 execution does not influence the user. The merging of two logical actions, DatReq and sDR, into a single 'physical' transition embodies a rather natural idea that there is no need to separate these two actions executed at two different access points.

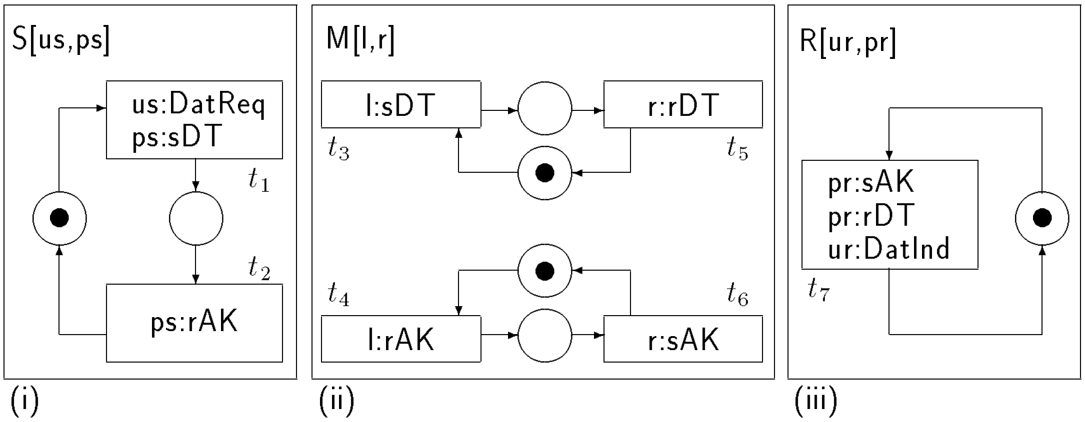

Fig. 4. Entities used in the example protocol: sender (i), media (ii) and receiver (iii) where $\mathsf{Id}_{\mathsf{ps}} = \mathsf{Id}_{\mathsf{l}}$ and $\mathsf{Id}_{\mathsf{pr}} = \mathsf{Id}_{\mathsf{r}}$.

The transfer media is represented by entity M (see Figure 4(ii)) with two access points, l and r, used respectively for the communication with its left and right entities; formally, this is specified by assuming that $\mathsf{Id}_{\mathsf{ps}} = \mathsf{Id}_{\mathsf{l}}$ and $\mathsf{Id}_{\mathsf{r}} = \mathsf{Id}_{\mathsf{pr}}$. 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.

The receiver is represented by entity R with two access points, ur and pr, for the communication with its user and the transfer media (see Figure 4(iii)). 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. In other words, instead of having three separate transitions, as it would be necessary 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 correctness verification. Moreover, the resulting protocol specification will satisfy the service in Figure 2(ii).

B. Operations on entities

Concurrent composition

The main operation for manipulating entities is concurrent composition. Let $\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$ and $\mathsf{pne}' = \langle \Sigma', \mathsf{acc}' \rangle$ be two entities such that $\Sigma = \langle S, T, F, M_0 \rangle$ and $\Sigma' = \langle S', T', F', M_0' \rangle$ are disjoint nets. The concurrent composition of pne and pne' is

$$\mathsf{pne} \| \mathsf{pne}' = \langle S'', T'', F'', M_0'', \mathsf{acc}'' \rangle$$

where the different components of the tuple on the right hand side are defined in the following way (below $\mathsf{Id} = \mathsf{Id}_{\mathsf{pne}} \cap \mathsf{Id}_{\mathsf{pne}'}$).

- The places S'' and the initial marking M_0'' are given by

$$S'' = S \cup S' \quad \text{and} \quad M_0'' = M_0 \cup M_0'.$$

- The set of transitions T'' is

$$T'' \;=\; \{t \in T \mid \bigoplus_{\mathsf{id} \in \mathsf{Id}} \mathsf{pne}_{\mathsf{id}}(t) = \emptyset_{\mathsf{m}}\} \cup \{t \in T' \mid \bigoplus_{\mathsf{id} \in \mathsf{Id}} \mathsf{pne}'_{\mathsf{id}}(t) = \emptyset_{\mathsf{m}}\} \cup T^{\mathsf{new}}$$

where T^{new} is the set of new transitions t_{QR} such that $Q \in \mathsf{Ind}_\Sigma$ and $R \in \mathsf{Ind}_{\Sigma'}$ are minimal sets satisfying, for every $\mathsf{id} \in \mathsf{Id}$,

$$\bigoplus_{t \in Q} \mathsf{pne}_{\mathsf{id}}(t) \quad = \quad \bigoplus_{t \in R} \mathsf{pne}'_{\mathsf{id}}(t).$$

- The pre-sets and post-sets remain unchanged for the transitions in $T'' \setminus T^{\mathsf{new}}$. For each $t_{QR} \in T^{\mathsf{new}}$, the pre-set is ${}^\bullet Q \cup {}^\bullet R$ and the post-set is $Q^\bullet \cup R^\bullet$, i.e., each t_{QR} inherits the connectivity of the transitions in Q and R.
- The set of access points acc'' is given by

$$\mathsf{acc}'' \quad = \quad \{\hat{\mathsf{c}} \mid \mathsf{c} \in (\mathsf{acc} \cup \mathsf{acc}') \wedge \mathsf{Id}_{\mathsf{c}} \notin \mathsf{Id}\}$$

where for each $\mathsf{c} \in \mathsf{acc} \cap \mathsf{acc}''$, $\hat{\mathsf{c}}$ is an access point defined thus

$$\hat{\mathsf{c}}(t) = \begin{cases} \mathsf{c}(t) & \text{if} \quad t \in T'' \cap T \\ \bigoplus_{u \in Q} \mathsf{c}(u) & \text{if} \quad t = t_{QR} \in T^{\mathsf{new}} \\ \emptyset_{\mathsf{m}} & \text{if} \quad t \in T'' \cap T'. \end{cases}$$

For $\mathsf{c} \in \mathsf{acc}' \cap \mathsf{acc}''$, $\hat{\mathsf{c}}$ is defined in a similar way.

The operation of concurrent composition is well defined, i.e., one can show that it always yields an entity (see the Appendix). It is also commutative and associative (up to net isomorphism).

Informally speaking, the concurrent composition of two entities, pne and pne', is a new entity which is the union of the Petri nets underlying pne and pne' such that transitions visible through the access points of the same sort in exactly the same way are synchronised. The visibility of the new transitions with respect to the remaining access points (denoted by $\hat{\mathsf{c}}$) is defined as the multiset sum of the labellings of the constituent transitions. All the transitions which were visible through one of the access points used for synchronisation are removed. The access points of the resulting entity is the union of the (suitably adjusted) original access points but except those which took place in the composition.

In Figure 3, the composition operation is indicated by connecting the corresponding access points. Figure 5 shows the result of performing the composition operation on the three entities from Figure 4. The resulting entity, $\mathsf{TE}[\mathsf{su}, \mathsf{sr}] = \mathsf{S} \| \mathsf{M} \| \mathsf{R}$, has two access points for protocol users, us and ur. Note that in applying the composition operation, transition t_1 was synchronised with t_3; t_2 with t_4; and t_5 and t_6 with t_7.

Hiding

When solving a specific tasks of protocol engineering, some access points may happen to be redundant. In such a case, the operation of hiding can be applied which removes some access points, but does not change the underlying net and, consequently, its behavior at the remaining access points. Let $\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$ be a Petri net entity and $H \subseteq \mathsf{acc}$ be a set of access points. Then $\mathsf{hide}_H(\mathsf{pne}) = \langle \Sigma, \mathsf{acc} \setminus H \rangle$. Clearly, $\mathsf{hide}_H(\mathsf{pne})$ is an entity.

The possibility of representing entities and operations on them both in schematical 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 3. At the lower level, each entity can be specified using a Petri net, as shown in Figure 4. The explicit representation of protocol architecture and the possibility of using different styles of specification [73] enhances the quality of specifications, making them easier to deal with.

Note that in the specification done at the system level, the operation of concurrent composition is only indicated. Its actual execution, resulting in the composition of the underlying Petri nets, may be needed for dealing with other tasks of protocol engineering such as analysis, verification and implementation. For instance, the analysis of the example protocol against its logical correctness specification (e.g., absence of deadlocks) might involve deriving the entity $\mathsf{hide}_{\{\mathsf{ur},\mathsf{us}\}}(\mathsf{S}\|\mathsf{M}\|\mathsf{R})$ which is nothing but a Petri net with no access points, and testing it for the presence of deadlocks.

It is worth noting that the Petri net in Figure 5 is more compact than that depicted in Figure 2. The same can be said about the corresponding reachability graphs. This is a direct consequence of allowing multiset labelling of transitions which can reduce the size of nets representing protocol systems.

C. Step sequence semantics and invisible transitions

To define a bisimulation equivalence for Petri net entities, we need to make precise what it means for a step to be executed in between two 'invisible' step sequences. Let

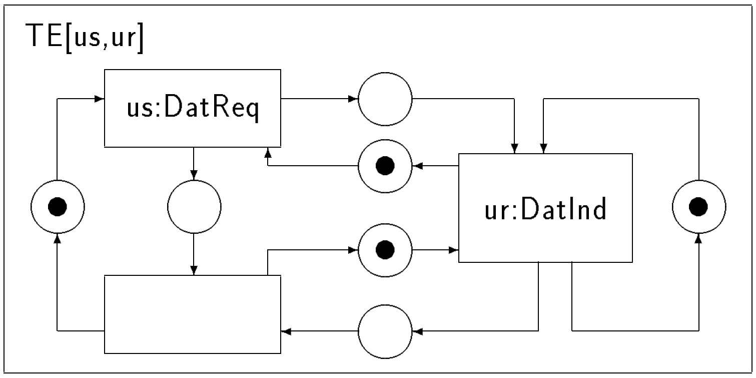

Fig. 5. Composition of entities

$\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$ be a Petri net entity. We first extend the notion of an access point to a set of transitions and a set of access points, i.e., for every non-empty set $U \subseteq T$ and $\mathsf{a} \in \mathsf{acc}$,

$$\mathsf{a}(U) = \bigoplus_{t \in U} \mathsf{a}(t) \quad \text{and} \quad \mathsf{acc}(U) = \bigoplus_{\mathsf{a} \in \mathsf{acc}} \mathsf{a}(U).$$

Note that if $\mathsf{a}(U) = \emptyset_{\mathsf{m}}$ then U is a set of transitions which are invisible from the access point a, and if $\mathsf{acc}(U) = \emptyset_{\mathsf{m}}$ then U is a set of transitions which are invisible from all the access points of the entity pne. As in CCS, we will adopt the view that invisible transitions can be disregarded when one considers the communication capability of a system, in this case, the Petri entity pne. In our framework, this leads to the definition of two auxiliary reachability relations, $\Longrightarrow$ and $\stackrel{U}{\Longrightarrow}$.

Let M and M' be two markings reachable from the initial marking of the entity pne. We will denote $M \Longrightarrow M'$ if there is a step sequence $\omega = U_1 \ldots U_k$ $(k \geq 0)$ such that $M\ [\omega\rangle\ M'$ and $\mathsf{acc}(U_1) \oplus \cdots \oplus \mathsf{acc}(U_k) = \emptyset_{\mathsf{m}}$. Moreover, if $U \subseteq T$ is a non-empty set of transitions such that

$$M \Longrightarrow M_1\ [U\rangle\ M_1' \Longrightarrow M$$

for some markings M_1 and M_1', then we denote $M \stackrel{U}{\Longrightarrow} M'$.

Intuitively, $M \Longrightarrow M'$ means that M can be transformed into M' without executing any transition which is externally visible, whereas $M \stackrel{U}{\Longrightarrow} M'$ means that starting at M one can execute step U possibly preceded and followed by a step sequence made up of invisible transitions. The main idea behind bisimulation equivalence is that as far as the external environment is concerned, $M \xRightarrow{U} M'$ and $M\ [U\rangle\ M'$ are equivalent executions.

D. Equivalence of entities

For dealing with the general problem of protocol verification, which usually amounts to demonstrating a correspondence between a protocol and its service, we need a suitable notion of equivalence allowing us to compare the behaviour of different entities. The definition below is based on the notion of an observational equivalence originally introduced for CCS [53].

Let $\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$ and $\mathsf{pne}' = \langle \Sigma', \mathsf{acc}' \rangle$ be two entities with the same sort and M_0 and M_0' be respectively the initial markings of Σ and Σ'. Then pne and pne' are bisimulation equivalent if there exists a relation $\mathsf{bis} \subseteq [M_0\rangle \times [M_0'\rangle$ such that $(M_0, M_0') \in \mathsf{bis}$ and if $(M_1, M_1') \in \mathsf{bis}$ then the following hold (below $\mathsf{Id} = \mathsf{Id}_{\mathsf{pne}} = \mathsf{Id}_{\mathsf{pne}'}$):

- If $M_1 \Longrightarrow M_2$ then there is M_2' such that $M_1' \Longrightarrow M_2'$ and $(M_2, M_2') \in \mathsf{bis}$.
- If $M_1\ [U\rangle\ M_2$ and $\mathsf{acc}(U) \neq \emptyset_{\mathsf{m}}$ then there are U' and M_2' such that $M_1' \xRightarrow{U'} M_2'$, $(M_2, M_2') \in \mathsf{bis}$ and for all $\mathsf{id} \in \mathsf{Id}$, $\mathsf{pne}_{\mathsf{id}}(U) = \mathsf{pne}'_{\mathsf{id}}(U')$.
- If $M_1' \Longrightarrow M_2'$ then there is M_2 such that $M_1 \Longrightarrow M_2$ and $(M_2, M_2') \in \mathsf{bis}$.
- If $M_1'\ [U'\rangle\ M_2'$ and $\mathsf{acc}'(U') \neq \emptyset_{\mathsf{m}}$ then there are U and M_2 such that $M_1 \xRightarrow{U} M_2$, $(M_2, M_2') \in \mathsf{bis}$ and for all $\mathsf{id} \in \mathsf{Id}$, $\mathsf{pne}_{\mathsf{id}}(U) = \mathsf{pne}'_{\mathsf{id}}(U')$.

We denote this by $\mathsf{pne} \approx \mathsf{pne}'$ or $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$, depending on the context. Note that the logical implementation of the example protocol shown in Figure 5 is bisimulation equivalent to its service shown in Figure 2(ii). The same is not true of the original specification in Figure 2(i). More precisely, if we assume that transitions t_2, t_3, t_4 and t_5 are invisible, then the net can execute $M_0 \xRightarrow{\{t_1\}} M_1 \xRightarrow{\{t_1\}} M_2$ which cannot be matched by any execution on the service side.

It is not difficult to show that $\approx$ is an equivalence relation in the usual sense. A crucial property, however, is that bisimulation equivalence is preserved by concurrent composition and hiding (see the Appendix).

Theorem IV.1: Let pne and pne' be protocol entities such that $\mathsf{pne} \approx \mathsf{pne}'$, and H and H' be sets of access points of respectively pne and pne' such that $\mathsf{Id}_H = \mathsf{Id}_{H'}$. Then $\mathsf{hide}_H(\mathsf{pne}) \approx \mathsf{hide}_{H'}(\mathsf{pne}')$. □

Theorem IV.2: Let pne, pne', pne_0 and pne_0' be protocol entities such that $\mathsf{pne} \approx \mathsf{pne}'$ and $\mathsf{pne}_0 \approx \mathsf{pne}_0'$. Then $(\mathsf{pne}\|\mathsf{pne}_0) \approx (\mathsf{pne}'\|\mathsf{pne}_0')$. □

The last result is essential for the bisimulation equivalence to be accepted as a valid notion of equivalence for the hierarchical verification of communication protocols.

E. Discussion

There are two important points to be made about the bisimulation equivalence we have just introduced. The first is that we do need a bisimulation equivalence based on the step sequence semantics, and cannot use a simpler notion of bisimulation based on interleaving semantics. Consider, for example, the entities in Figure 6. Clearly, pne_1 and pne_2 are interleaving bisimilar. However, if we compose each of them with pne, then the entities $\mathsf{pne}_1' = (\mathsf{pne}_1\|\mathsf{pne})$ and $\mathsf{pne}_2' = (\mathsf{pne}_2\|\mathsf{pne})$ are no longer equivalent since pne_2' is deadlocked, while pne_1' can execute a visible transition.

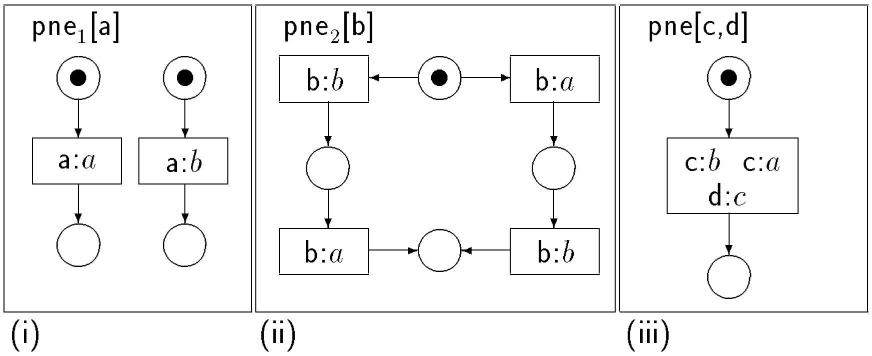

Fig. 6. Bisimulation based on interleaving is not preserved through composition, $\mathsf{Id_a} = \mathsf{Id_b} = \mathsf{Id_c}$.

The second point is that we cannot replace the current definition by a weaker one in which we would require that two entities be bisimulation equivalent, separately, for each pair of the corresponding access points. Again, we can show that this would lead to a loss of the preservation of behavioural equivalence through concurrent composition. For consider the entities pne and pne' in Figure 13(i,ii). We first observe that $\mathsf{hide}_{\{\mathsf{a}\}}(\mathsf{pne}) \approx \mathsf{hide}_{\{\mathsf{a}'\}}(\mathsf{pne}')$ and $\mathsf{hide}_{\{\mathsf{b}\}}(\mathsf{pne}) \approx \mathsf{hide}_{\{\mathsf{b}'\}}(\mathsf{pne}')$, i.e., pne and pne' would be bisimulation equivalent if $\approx$ were redefined in the way we just described. However, pne and pne' can hardly be regarded as equivalent entities. To show this we compose them separately with two other entities, pne_1 and pne_2, shown in Figure 13(iii,iv). The resulting entities, $\mathsf{pne}_1' = \mathsf{pne}_1 \| \mathsf{pne} \| \mathsf{pne}_2$ and $\mathsf{pne}_2' = \mathsf{pne}_1 \| \mathsf{pne}' \| \mathsf{pne}_2$, are presented in Figure 13(v,vi). It is easy to see that pne_1' and pne_2' have quite different behaviour; in particular, the former can execute some transitions, including that visible through access point g, whereas the latter is completely deadlocked. Note finally that, according to the definition we have chosen, pne_1 and pne_2 are *not* behaviourally equivalent, $\mathsf{pne}_1 \not\approx \mathsf{pne}_2$.

F. Correctness of hierarchical protocols

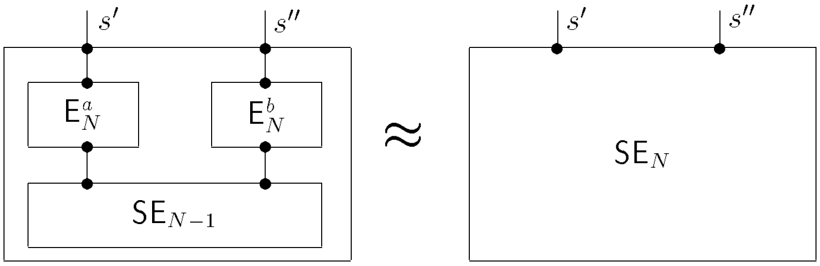

Fig. 7. Protocol verification.

Using the notion of bisimulation equivalence we can formally state the problem of the verification of a layered protocol. The verification of an N-layer protocol consists 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, E_N^a and E_N^b, with the entity of the (N-1)-service, SE_{N-1}, as shown in Figure 7. Assuming that we also have an entity which specifies the reference N-service, SE_N, the verification problem reduces to proving that SE_N is bisimulation equivalent to the composition of E_N^a, E_N^b and SE_{N-1}. The N-protocol which is specified by the entity $\mathsf{E}_N = \mathsf{E}_N^a \| \mathsf{E}_N^b$ is considered to be correct with respect to services SE_N and SE_{N-1} if such an equivalence holds.

The framework we have outlined allows one to model hierarchical composition of protocol layers in a rather straightforward way (see Figure 8). Indeed, if $\mathsf{E}_N[s_{N-1}, s_N] = \mathsf{E}_N^a \| \mathsf{E}_N^b$ and $\mathsf{E}_{N+1}[s_N', s_{N+1}] = \mathsf{E}_{N+1}^a \| \mathsf{E}_{N+1}^b$ are respectively the protocol entities of the N- and (N+1)-layers and $\mathsf{Id}_{s_N} = \mathsf{Id}_{s_N'}$, then the entity that corresponds to their hierarchical composition is $\mathsf{E}_H = (\mathsf{E}_N \| \mathsf{E}_{N+1})$. The theorem below follows immediately from the commutativity and associativity of concurrent composition and the fact that bisimulation equivalence is preserved through concurrent composition.

Theorem IV.3 (Hierarchical composition) If the N-protocol defined as entity E_N is correct with respect to services SE_N and SE_{N-1} and the (N+1)-protocol defined as entity E_{N+1} is correct with respect to services SE_{N+1} and SE_N then the hierarchical protocol E_H is also correct with respect to services SE_{N+1} and SE_{N-1} (see Figure 8). $\square$

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

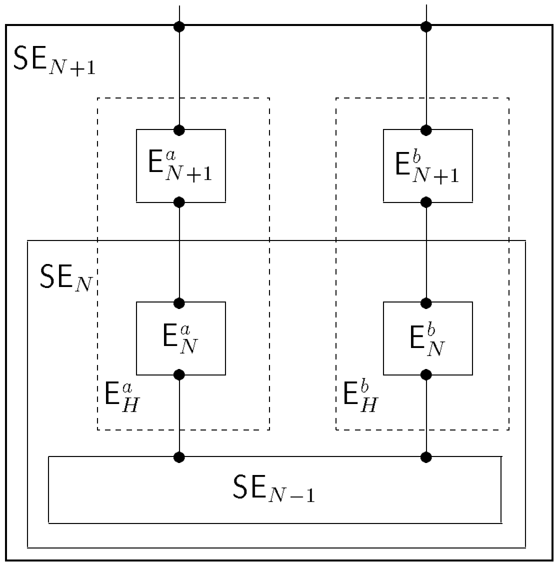

Fig. 8. Protocol hierarchy.

V. Composition rules for the entity level

A protocol entity used at the system level may itself have complex internal structure; in practice, it will be described as a set of procedures combined together by means of specific composition rules.

The interface of a procedure will be expressed in two ways. Firstly, a procedure should have a potential to communicate with other entities similarly as the enclosing entity and, therefore, it should have the same interface in terms access points. Secondly, to allow joining procedures into entities, some composition rules are needed and this requires some additional information about the states of the procedures. For instance, to perform sequential composition of two procedures, we need to have their initial and final states and, for the disabling operation, an information about the reachable states of the disabled procedure. Such an information will be formally captured by the notion of a macrostate, which is a set of markings of the Petri net underlying a protocol procedure [2], [5].

A macrostate of a Petri net N is a non-empty set of its non-empty 1-safe markings. The base $\lfloor \mathsf{u} \rfloor$ of a macrostate u is defined as $\lfloor \mathsf{u} \rfloor = \bigcup_{M \in \mathsf{u}} M$. A macrostate u is unitary if it contains only one marking, $\mathsf{u} = \{M\}$. Moreover, if M is a marking under which exactly one place is marked, then u is called simple.

A. Protocol procedures

A protocol procedure is a tuple $\mathsf{Proc} = \langle N, \mathsf{acc}, \mathsf{head}, \mathsf{tail}, \mathsf{reach} \rangle$ where $N = \langle S, T, F \rangle$ is a Petri net, acc is finite set of access points for N, and head, tail and reach are macrostates for N, called respectively the head, tail and reachable macrostates of Proc. As in the case of the definition of an entity, we assume that distinct access points in Proc have distinct sorts, and denote $\mathsf{Id}_{\mathsf{Proc}} = \mathsf{Id}_{\mathsf{acc}}$. There are two properties required of the sets of macrostates, namely for every $M \in \mathsf{head}$,

- $[M\rangle$ is a set of 1-safe markings included in reach.
- If M' and M'' are markings in $\mathsf{tail} \cup [M\rangle$ and $M' \subseteq M''$ then $M' = M''$.

An intuitive meaning of head and tail is that they comprise the initial and final markings of Proc. The third macrostate, reach, contains all possible markings reachable from the markings in head (although there may be markings in reach which will never be reached from any of the markings in head). Thus, a protocol procedure is essentially an entity without the explicit initial marking, but instead with three sets of states carrying information needed later for applying composition operators.

In the diagrams, protocol procedures will be represented as entities with additional information specifying their states. If the head (or tail) state contains only one marking M, the places marked at M will be indicated by incoming (viz. outgoing) short arcs.

The sender and receiver entities of the protocol in Figure 4 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 9, 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 all specified using just one transition.

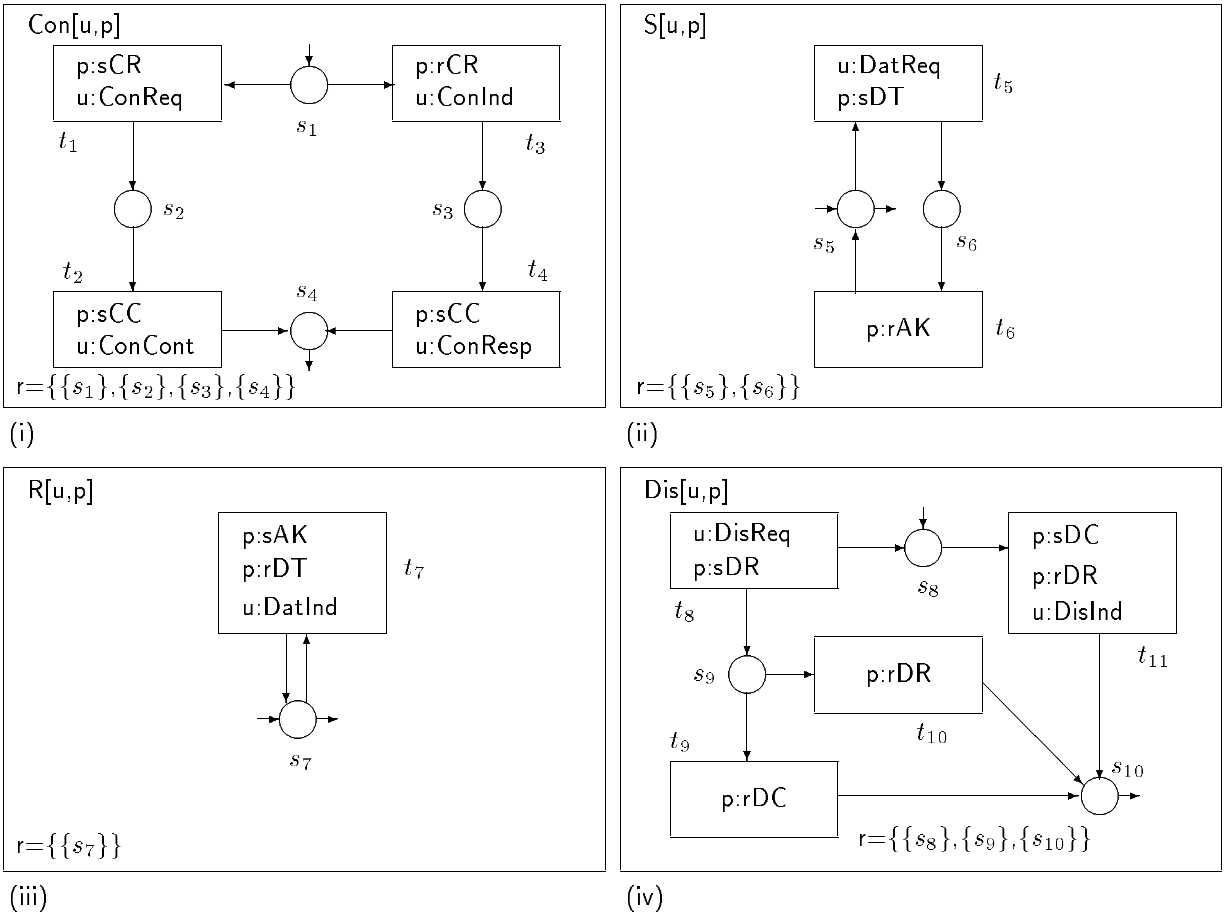

Fig. 9. Protocol procedures: connection establishment (i), sending data (ii), receiving data (iii) and disconnection (iv).

B. Operations on protocol procedures

To obtain a complex entity from a set of protocol procedures we use six operations: transformation into an entity, sequential and choice compositions, iteration, disabling and parallel composition. The last one is defined thus.

Parallel composition

This operation simply juxtaposes the nets of two procedures. The macrostates are formed as all possible combinations of markings of the corresponding macrostates of the two procedures. Formally, the parallel composition of two protocol procedures with the same sort, Proc and Proc' is:

$$\mathsf{Proc}|||\mathsf{Proc}' = \langle N \oplus N', \mathsf{acc} \oplus \mathsf{acc}', \mathsf{head} \oplus \mathsf{head}', \mathsf{tail} \oplus \mathsf{tail}', \mathsf{reach} \oplus \mathsf{reach}' \rangle$$

where $N \oplus N'$ is the disjoint union of N and N'; for two macrostates, u and u', we denote $\mathsf{u} \oplus \mathsf{v} = \{M \cup M' \mid M \in \mathsf{u} \wedge M' \in \mathsf{u}'\}$; and $\mathsf{acc} \oplus \mathsf{acc}'$ is a set of access points with the same sort as Proc and Proc', defined thus. For every $\mathsf{id} \in \mathsf{Id}_{\mathsf{Proc}} = \mathsf{Id}'_{\mathsf{Proc}}$, there is a unique $\mathsf{a} \in \mathsf{acc} \oplus \mathsf{acc}'$ such that $\mathsf{Id}_{\mathsf{a}} = \mathsf{id}$ and for every transition t coming from Proc (from Proc'), $\mathsf{a}(t) = \mathsf{acc}_{\mathsf{id}}(t)$ (resp. $\mathsf{a}(t) = \mathsf{acc}'_{\mathsf{id}}(t)$).

Defining the remaining four composition operations is more complicated. We discuss this using sequential composition as an example.

An intuitive meaning of a sequential composition of two protocol procedures, Proc and Proc', is that the result, denoted by $\mathsf{Proc};\mathsf{Proc}'$, should be a procedure which first behaves like Proc and after terminating in one of the valid final markings of Proc, it can further proceed as if it were Proc' starting in one of its valid initial markings. Such a description of the meaning of the sequential composition should be matched by an appropriate definition on the net level. Indeed, there are several examples in the literature where this has been done [24], [45], [57], [70]. However, one could face a problem with the definition of a single operator for all possible combinations of Proc and Proc'. Basically, there can be nets which define behaviours in too abstract a way. Consider the protocol procedures in Figure 10. Proc is composed of two completely independent components and has a single initial and final marking, while Proc' can begin its operation in one of two different markings and terminate likewise. If we now follow the idea behind the sequential composition then somehow, upon a successful termination, the two components of Proc should decide to pass the control to the same initial marking of Proc'. But this would imply that there should be some sort of interaction between them, which would contradict the given specification of Proc. The situation we have just described can be thought of as an example that the description of Proc is just too abstract for the purpose of composing the two procedures. In particular, one should include some additional information about the synchronisation between the two components of Proc in order to jointly select exactly one of the two possible initial markings of Proc′. At this point one can observe that if Proc′ had only one initial (head) marking, then there would be *no need* to synchronise the two components of Proc. This is the main reason for assuming from now on that the head macrostates are always unitary.

Another situation which can lead to problems is the possibility of 'going back' to Proc after having passed the control to Proc′. We will prevent this by putting a restriction on the arcs outgoing from (incoming to) places in the tail (head) markings of Proc (Proc′).

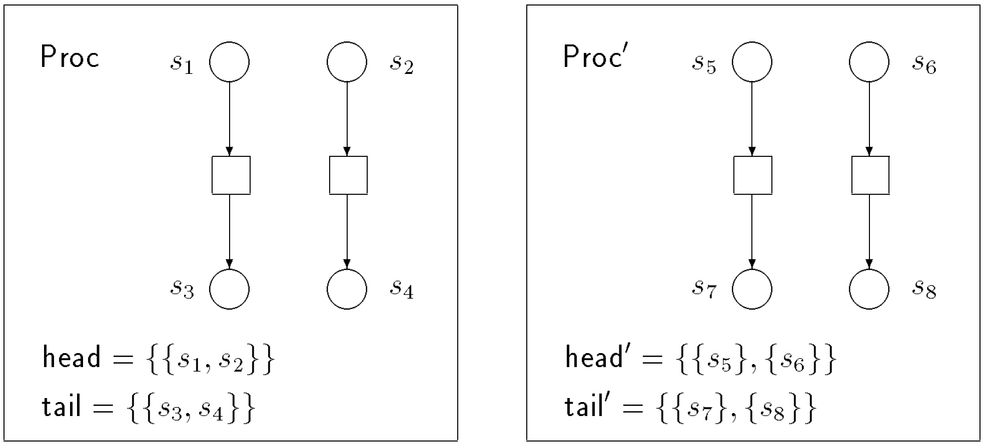

Fig. 10. Sequential composition is not always directly applicable.

Following the above observations, we will define sequential composition as an operation restricted to well-behaved cases, characterised by simple conditions imposed on the tail macrostate of Proc and the head macrostate of Proc′. This should not be considered as a major restriction because there are simple behaviour preserving transformation rules for protocol procedures (such as one step unwinding) allowing one to bypass them in practice. The same remark applies to the choice composition, iteration and disabling. Before giving their formal definition, we present an auxiliary operation of macrostate merge.

Macrostate merge

Let $N = \langle S, T, F\rangle$ be a Petri net, and v and w be two macrostates of N such that $\lfloor \mathsf{v} \rfloor \cap \lfloor \mathsf{w} \rfloor = \emptyset$ and ${}^\bullet\lfloor \mathsf{v} \rfloor \cap {}^\bullet\lfloor \mathsf{w} \rfloor = \emptyset = \lfloor \mathsf{v} \rfloor^\bullet \cap \lfloor \mathsf{w} \rfloor^\bullet$. Merging v and w yields a new net $N' = \langle S', T', F'\rangle$, defined thus.

- The place set S' is defined as[3]

$$S' \ = \ S \ \setminus \ (\lfloor \mathsf{v} \rfloor \cup \lfloor \mathsf{w} \rfloor) \ \cup \ (\lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor).$$

- The transition set T' is defined as

$$T' \ = \ T^0 \ \cup \ T^{\mathsf{v}} \ \cup \ T^{\mathsf{w}} \ \cup \ T^{\mathsf{vw}}$$

  where

$$\begin{array}{rrcll} & T^0 & = & \{t \mid & t \in T \ \wedge \ \mathsf{v}_t = \emptyset = \mathsf{w}_t\} \\ & T^{\mathsf{v}} & = & \{t_M \mid & t \in T \ \wedge \ \mathsf{v}_t \neq \emptyset = \mathsf{w}_t \ \wedge \ M \in \mathsf{w}\} \\ & T^{\mathsf{w}} & = & \{t_L \mid & t \in T \ \wedge \ \mathsf{v}_t = \emptyset \neq \mathsf{w}_t \ \wedge \ L \in \mathsf{v}\} \\ \text{and} & T^{\mathsf{vw}} & = & \{t_{ML} \mid & t \in T \ \wedge \ \mathsf{v}_t \neq \emptyset \neq \mathsf{w}_t \ \wedge \ M \in \mathsf{w} \ \wedge \ L \in \mathsf{v}\} \end{array}$$

  and we use u_t to denote $({}^\bullet t \cup t^\bullet) \cap \lfloor \mathsf{u} \rfloor$, for $\mathsf{u} \in \{\mathsf{v}, \mathsf{w}\}$.

- The flow relation F' of N' is defined in the following way (we only give the definition for the arcs from places to transitions, those pointing in the opposite direction are defined similarly).
  - For a place $s \in S' \cap S$ and a transition z in one of the forms: $z = t \in T^0$, $z = t_M \in T^{\mathsf{v}}$, $z = t_L \in T^{\mathsf{w}}$ and $z = t_{ML} \in T^{\mathsf{vw}}$, we have $(s, z) \in F'$ if and only if $(s, t) \in F$.
  - For a place $s = (p, q) \in \lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor$ and a transition $t \in T^0$, $(s, t) \notin F'$.
  - For a place $s = (p, q) \in \lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor$ and a transition $z = t_M \in T^{\mathsf{v}}$ (or $z = t_L \in T^{\mathsf{w}}$), $(s, z) \in F'$ if and only if $(p, t) \in F$ and $q \in M$ (resp. $(q, t) \in F$ and $p \in L$).
  - For a place $s = (p, q) \in \lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor$ and a transition $z = t_{ML} \in T^{\mathsf{vw}}$, $(s, z) \in F'$ if and only if either $(p, t) \in F$ and $q \in M$, or $(q, t) \in F$ and $p \in L$.[4]

The net N' will be denoted by $\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(N)$. Intuitively, in N', upon reaching any marking in one of the macrostates, v and w, the net may continue as though it were restarted in

[3] We assume that $\lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor$ are 'fresh' places, i.e., $(\lfloor \mathsf{v} \rfloor \times \lfloor w \rfloor) \cap S = \emptyset$.

[4] Note that since ${}^\bullet\lfloor \mathsf{v} \rfloor \cap {}^\bullet\lfloor \mathsf{w} \rfloor = \emptyset = \lfloor \mathsf{v} \rfloor^\bullet \cap \lfloor \mathsf{w} \rfloor^\bullet$, it is never the case that $(p, t) \in F$ and $(q, t) \in F$.

any of the markings of the other macrostate. In constructing N', the places marked under the two macrostates are 'multiplied' and the transitions adjacent the them 'split'. Figure 11 illustrates the application of the macrostate merge operation.

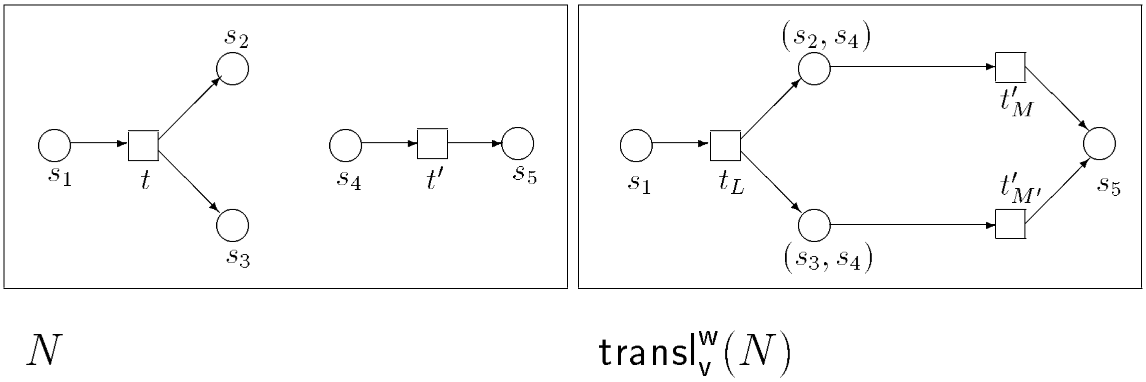

Fig. 11. Applying macrostate merge with $\mathsf{v} = \{M, M'\} = \{\{s_2\}, \{s_3\}\}$ and $\mathsf{w} = \{L\} = \{\{s_4\}\}$.

To be able to apply macrostate merge to protocol procedures, we need to describe its impact on other components of a procedure, viz. the macrostates and access points. First, for every macrostate u of N, we define $\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{u}) = \{\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(M) \mid M \in \mathsf{u}\}$ where, for every 1-safe marking M,

$$\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(M) \;=\; (M \cap S') \cup \{(p,q) \in \lfloor \mathsf{v} \rfloor \times \lfloor \mathsf{w} \rfloor \mid \{p,q\} \cap M \neq \emptyset\}.$$

Second, for an access point a of N, $\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{a})$ is defined to be an access point with the same sort as a and such that $\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{a})(z) = \mathsf{a}(t)$, for every $z = t \in T^0$, $z = t_M \in T^{\mathsf{v}}$, $z = t_L \in T^{\mathsf{w}}$ and $z = t_{ML} \in T^{\mathsf{vw}}$. Then, for every set acc of access points of N,

$$\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{acc}) = \{\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{a}) \mid \mathsf{a} \in \mathsf{acc}\}.$$

Finally, by bringing the last three definition together, for every protocol procedure

$$\mathsf{Proc} = \langle N, \mathsf{acc}, \mathsf{head}, \mathsf{tail}, \mathsf{reach} \rangle$$

and two macrostates, v and w, as above, we define $\mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{Proc})$ to be the following tuple:

$$\langle \mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(N), \mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{acc}), \mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{head}), \mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{tail}), \mathsf{transl}_{\mathsf{v}}^{\mathsf{w}}(\mathsf{reach}) \rangle.$$

We now can define the four remaining composition operations on protocol procedures, assuming that $\mathsf{Proc} = \langle N, \mathsf{acc}, \mathsf{head}, \mathsf{tail}, \mathsf{reach} \rangle$ and $\mathsf{Proc}' = \langle N', \mathsf{acc}', \mathsf{head}', \mathsf{tail}', \mathsf{reach}' \rangle$

are two protocol procedures with the same sorts and disjoint nets. Recall that the head macrostates of the two protocol procedures are unitary, i.e., they always contain exactly one marking.

Sequential composition

This operation combines two procedures by merging the tail macrostate of Proc with the head macrostate of Proc'. The head macrostate of the result comes from the first procedure, and the tail macrostate from the second one. Let $\lfloor \mathsf{tail} \rfloor^\bullet = \emptyset$ or ${}^\bullet\lfloor \mathsf{head}' \rfloor = \emptyset$. Then the sequential composition of Proc and Proc' is

$$\mathsf{Proc}; \mathsf{Proc}' = \mathsf{transl}_{\mathsf{tail}}^{\mathsf{head}'}(\langle N \oplus N', \mathsf{acc} \oplus \mathsf{acc}', \mathsf{head}, \mathsf{tail}', \mathsf{reach} \cup \mathsf{reach}' \rangle).$$

Iteration

Iteration simply merges the head and tail macrostates. Let head and tail be simple macrostates and $\lfloor \mathsf{tail} \rfloor^\bullet = \emptyset = {}^\bullet\lfloor \mathsf{head} \rfloor$. Then the iteration of Proc is

$$*(\mathsf{Proc}) = \mathsf{transl}_{\mathsf{head}}^{\mathsf{tail}}(\mathsf{Proc}).$$

Choice composition

This operation combines two procedures by merging their head macrostates. Let ${}^\bullet\lfloor \mathsf{head} \rfloor = \emptyset$ and ${}^\bullet\lfloor \mathsf{head}' \rfloor = \emptyset$. Then the choice composition of Proc and Proc' is

$$\mathsf{Proc} \Box \mathsf{Proc}' = \mathsf{transl}_{\mathsf{head}}^{\mathsf{head}'}(\langle N \oplus N', \mathsf{acc} \oplus \mathsf{acc}', \mathsf{head} \cup \mathsf{head}', \mathsf{tail} \cup \mathsf{tail}', \mathsf{reach} \cup \mathsf{reach}' \rangle).$$

Disabling

In this operation, the reachable macrostate of the first net is merged with the head macrostate of the second one. As a result, Proc' can start its execution at any time no matter what marking Proc is currently in. Let head' be simple and ${}^\bullet\lfloor \mathsf{head}' \rfloor = \emptyset$. Then the disabling of Proc by Proc' is

$$\mathsf{Proc} \; [\rangle \; \mathsf{Proc}' = \mathsf{transl}_{\mathsf{reach}}^{\mathsf{head}'}(\langle N \oplus N', \mathsf{acc} \oplus \mathsf{acc}', \mathsf{head}, \mathsf{tail} \cup \mathsf{tail}', \mathsf{reach} \cup \mathsf{reach}' \rangle).$$

We then present the last operation defined for protocol procedures.

Procedure-to-entity transformation

The *procedure-to-entity* operation takes $\mathsf{Proc} = \langle N, \mathsf{acc}, \mathsf{head}, \mathsf{tail}, \mathsf{reach}\rangle$ and produces an entity

$$\mathsf{Entity}(\mathsf{Proc}) = \langle N, \mathsf{acc}, M\rangle$$

where M is the only marking in head. This operation is needed for turning protocol procedures into protocol entities.

Example

In Figure 12, we show the result of applying the compositional rules to the procedures from Figure 9. We can illustrate the relationship between the specifications in Figures 9 and 12 using the executions of different transition sequences.

- The sequence $\{t_1\}\{t_2\}\{t_5\}\{t_6\}\{t_7\}\{t_8^4\}\{t_9\}$ corresponds to normal execution including connection establishment initiated by the entity PE ($\{t_1\}\{t_2\}$), sending of data block ($\{t_5\}\{t_6\}$), receiving data block ($\{t_7\}$) and disconnection ($\{t_8^4\}\{t_9\}$) initiated by the entity.
- The sequence $\{t_3\}\{t_4\}\{t_5\}\{t_{11}^5\}$ corresponds to connection establishment initiated by the peer entity ($\{t_3\}\{t_4\}$), the beginning of sending data block ($\{t_5\}$) and abnormal disconnection initiated by the peer entity ($\{t_{11}^5\}$).
- The sequence $\{t_3\}\{t_{11}^2\}$ corresponds to receiving a connection request with a simultaneous delivering to user ($\{t_3\}$) and sending a disconnection ($\{t_{11}^2\}$) that indicates connection rejection.
- The sequence $\{t_8^1\}\{t_9\}$ corresponds to sending a disconnection request in the initial state ($\{t_8^1\}$) with receiving a confirmation ($\{t_9\}$).

Such sequences are allowed in many protocols.

C. Behavioral equivalence of procedures

The notion of behavioural equivalence of two procedures is defined as in the case of protocol entities with one condition added, namely we allow only the markings in head (tail) macrostates to be in bisimulation relation with head (tail) macrostates. Reusing the already defined notion of bisimulation equivalence, we say that two protocol procedures

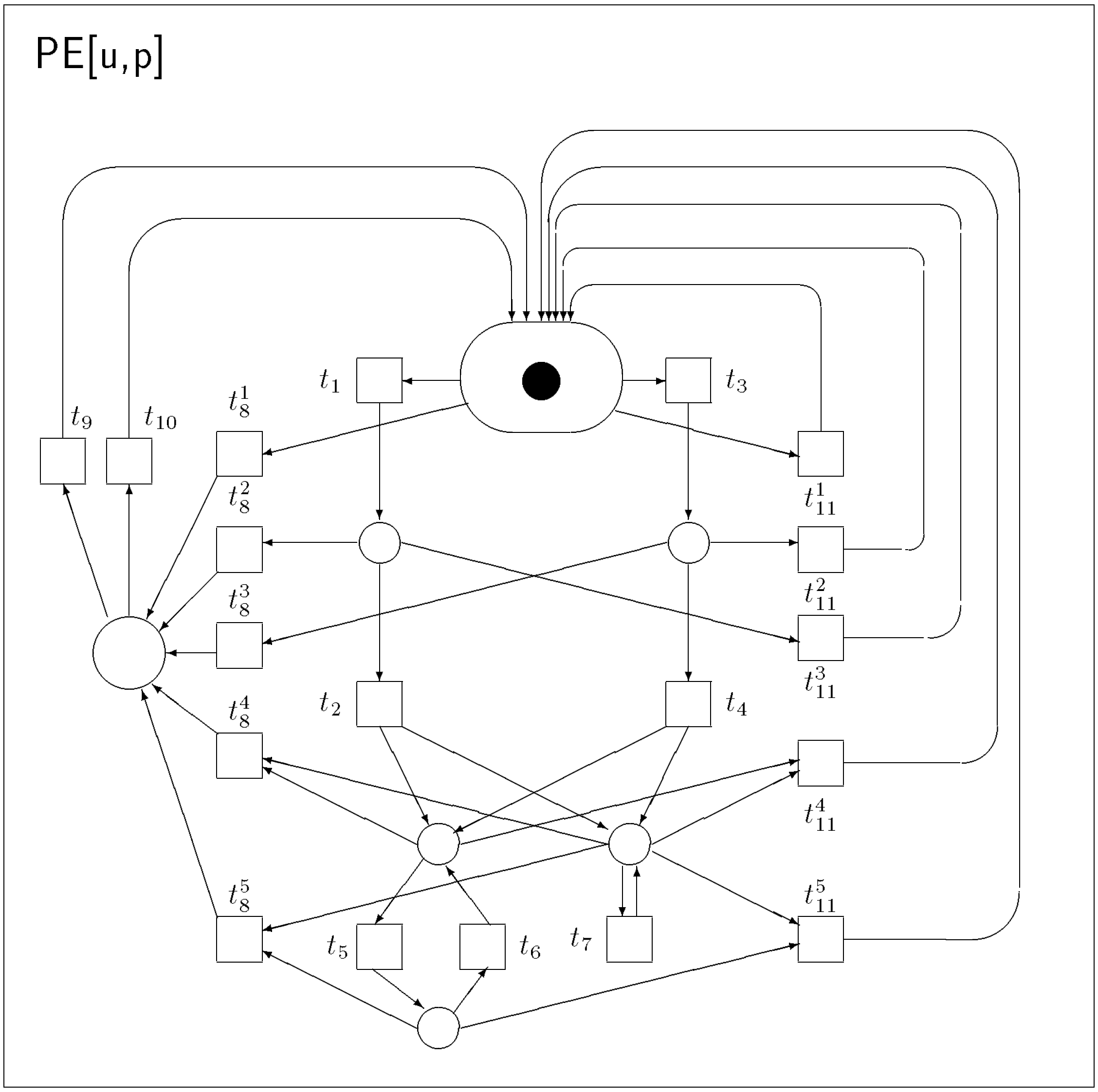

Fig. 12. Composition of protocol procedures: $\mathsf{Entity}(*((\mathsf{Con}\ ;\ (\mathsf{S}\ |||\ \mathsf{R}))\ [\rangle\ \mathsf{Dis}))$.

with the same sort, Proc and Proc', are bisimulation equivalent if there is a relation bis such that $\mathsf{Entity}(\mathsf{Proc}) \approx_{\mathsf{bis}} \mathsf{Entity}(\mathsf{Proc}')$ and, for every $(M, M') \in \mathsf{bis}$, $M \in \mathsf{head} \Leftrightarrow M' \in \mathsf{head}'$, and $M \in \mathsf{tail} \Leftrightarrow M' \in \mathsf{tail}'$. One can then see that bisimulation equivalence is preserved by all the operations defined for procedures in this section.

VI. Procedure Level

The main role of the procedure level is the development of correct protocol procedures which could be used as building blocks at other levels of compositionality. Protocol procedures are the result of a functional decomposition of protocols. What we mean by this is that by starting from a complex protocol design, one tries to break it down onto as small as possible logically indivisible units (we will call them *elementary procedures*) which then can be used in the compositional verification of the original complex design. This means that protocol procedures will not, in general, be designed using entirely compositional approach, i.e., the designer would first face the task of developing the necessary elementary procedures and after that, using them as the basic building blocks prove the correctness of the overall design. It is therefore important that the existing methods and results on protocol specification and verification can be employed at the procedure level of compositionality, since they are typically oriented towards and effective for protocols of relatively small size. We can identify at least four different ways in which theory and experience in this area can be utilised:

- A direct verification that consists in the specification of a procedure and its service in terms of Petri nets and checking whether they are bisimulation equivalent. A possible way to carry out the verification may be to generate reduced reachability graphs and then use efficient verification techniques, e.g., see [44], [72].
- Selective behavioural verification, where one constructs a procedure specification and checks it for some general properties such as deadlock-freeness, liveness and boundedness. In this case one can apply techniques such as model checking and partial order verification, e.g., see [39], [58], [68].
- Import well-defined procedures from other formalisms where their correctness was formally proved. In this case one has to ensure that the transformation mechanism from one model to another is correct with respect to behavioural properties. Notable examples of such transformations are [3], [10], [57], [70].
- Use procedures whose correctness have not be proved formally, but which have proved themselves by informal techniques like simulation and long term practical exploitation.

VII. Compositionality of Behaviour

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.[5] It is worth pointing out in this context that the technique of labelling transitions with 'simultaneous' actions can substantially 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 for the design of protocol systems must enjoy clearly defined and well-understood behavioural properties. For example, suppose that one has defined a protocol procedure Proc as a choice composition of two procedures, Proc' and Proc''. Then it should be expected that each of the possible behaviours of Proc be either a valid behaviour of Proc' or a valid behaviour of Proc''. In essence, what one should require of the operators used in compositional setting is that the behaviour of the resulting system should be a composition of the behaviours of the constituent subsystems (i.e., their union in the case of choice composition, a suitably defined concatenation in the case of sequential composition, etc.).

The above, highly desirable, property has been investigated, in particular, in [18], [46], [47]. It turns out that one can formulate simple conditions about the operators on Petri nets which ensure compositionality of behaviour as well as structure. The idea behind such a property can be explained in the following way. Suppose that op is a well-designed net operator. Then one can show that for its valid application $N = \mathsf{op}(N_1, \ldots, N_k)$ the following hold:

- If T_i is a valid execution step for N_i $(i \in I)$ then the union of the T_i's is a valid execution step for N.
- If T is a valid execution step for N then T can be partitioned onto execution steps T_i $(i \in I)$ such that each T_i is a valid execution step of N_i.

[5] This problem is commonly referred to as the 'combinatorial state explosion'.

The above turns out to be a very strong property. To begin with, it allows one to describe the execution behaviour of $\mathsf{op}(N_1, \ldots, N_k)$ 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 [18], [46], [47]. What is more, it allows one to derive in an automatic way a process algebra, similar to CCS [53] or ACP [9], together with suitable inference rules and equivalences, which is operationally equivalent (with respect to bisimulation equivalence) to the compositionally defined nets. Thus there exists a close relationship between two models of concurrent behaviour, namely Petri nets and process algebras. In particular, different tools and verification techniques (such as reduction rules) developed for process algebras can be imported into the Petri net based framework. All the operators defined for protocol procedures in this paper support compositionality of behaviour in the above sense.

VIII. Petri Net Tool

The compositional approach to communication protocol design outlined in this paper is supported by a computer-aided tool, called PN^3-Tool, whose detailed description can be found in [4]. The tool has been developed under Microsoft Windows 3.x and runs on PC IBM compatible computers. The main functionality of PN^3-Tool is supported by three editors, namely the basic, algebraic and architectural editors:

- The basic editor allows one to directly produce specifications in Petri net form. It implements the standard Petri net editing facilities such as creating, removing and resizing Petri net elements (place, transitions, arcs, markings and labels).
- The algebraic editor is driven by a set of algebraic operators which can be used to define a Petri net specification. The operations which have been implemented include the sequential, parallel and choice compositions, iteration and disabling.
- At the top of the hierarchy is the architectural editor, where the user can manipulate Petri net entities in the diagrammatical form, i.e., in the form of interconnected boxes. Each Petri net entity may refer to another entity configuration (on another page) or directly to a specified Petri net. To compose two entities one only needs to connect the corresponding access points. The user can produce the specification both by top-down and bottom-up approach.

Some additional features have also been implemented. These include visual simulation of a Petri net which can used for better understanding of its dynamic behavior and analysis; verification of several important properties (e.g., boundedness and deadlock-freeness) by means of the Petri net coverability tree, and comparing two Petri nets for bisimulation equivalence.

IX. Conclusion

In this paper we outlined a systematic approach to the design of communication protocols. The approach is based on Petri nets as an underlying formal model and compositionality which is applied at various levels of development.

There are other issues in protocol engineering which we believe could be successfully addressed by extending the framework presented in this paper. The first one is manipulation on data as, for example, in [17]. Another issue is that of a linguistic specification environment for a Petri net based model, similar to LOTOS [23] or Estelle [26]. Some general results in this area already exist [3], [10], [17], but we feel that they need some modification in order to better fit the area of application. Also, it is important to investigate other compositional rules for procedures which preserve behavioural correctness. Some initial research in this are can be found in [29], [30], [75]. 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 [39], [72], could be improved by taking advantage of compositional definition of nets. Moreover, a strong link with process algebras should allow one to import process algebra specific verification techniques (such as axiomatisation of bisimulation equivalence and rewriting rules) into the Petri net based framework.

Appendix

I. Multisets

A multiset over the set A of elementary names of communication primitives is a mapping $\mathsf{m}\colon \mathsf{A} \to \{0, 1, 2, \ldots\}$. If $\mathsf{m}(a) = 0$, for all $a \in \mathsf{A}$, then m will be called the empty multiset and denoted by $\emptyset_{\mathsf{m}}$. The multiset is finite if $\mathsf{m}(a) > 0$ only for finitely many $a \in \mathsf{A}$. The set of finite multisets over A is denoted by $\mathsf{mult}(\mathsf{A})$. The usual set enumeration notation can readily be extended to multisets. For instance, $\{a, a, b, c, c, c\}$ denotes the multiset m such that $\mathsf{m}(a) = 2$, $\mathsf{m}(b) = 1$, $\mathsf{m}(c) = 3$ and $\mathsf{m}(x) = 0$ otherwise. The sum of multisets $\mathsf{m}_1, \ldots, \mathsf{m}_k$ over A is a multiset $\mathsf{m}_1 \oplus \cdots \oplus \mathsf{m}_k$ over A defined by

$$(\mathsf{m}_1 \oplus \cdots \oplus \mathsf{m}_k)(a) = \mathsf{m}_1(a) + \cdots + \mathsf{m}_k(a),$$

for every $a \in \mathsf{A}$. If $A = \{\mathsf{m}_1, \ldots, \mathsf{m}_k\} = \{\mathsf{m}(t) \mid t \in Q\}$ is an indexed multiset of multisets over A, then

$$\bigoplus_{t \in Q} \mathsf{m}(t) = \mathsf{m}_1 \oplus \cdots \oplus \mathsf{m}_k.$$

II. Basic properties of protocol entities

We first observe that the operation of concurrent composition of two entities always yields an entity. The only non-trivial property to check is the 1-safeness of the underlying net. This, however, follows immediately from the fact that the nets in the composed entities are disjoint, and that the new transitions inherit the full connectivity of sets of independent transitions from the two composed entities. As a result, the execution of a transition $t_{QR} \in T^{\mathsf{new}}$ has exactly the same effect (in terms of the marking which is generated) as the execution of the step $U = Q \cup R$ in the union of the nets underlying the two entities which, of course, is a 1-safe Petri net. It is also clear that hiding a set of access points always yields an entity since the underlying Petri net does not change.

That the concurrent composition of entities is a commutative operation,

$$\mathsf{pne}^1 \| \mathsf{pne}^2 = \mathsf{pne}^2 \| \mathsf{pne}^1,$$

follows directly from the definition of $\|$. The associativity property,

$$\mathsf{pne}^1 \| (\mathsf{pne}^2 \| \mathsf{pne}^3) = (\mathsf{pne}^1 \| \mathsf{pne}^2) \| \mathsf{pne}^3,$$

holds under the proviso that $\mathsf{Id}_{\mathsf{pne}^1} \cap \mathsf{Id}_{\mathsf{pne}^2} \cap \mathsf{Id}_{\mathsf{pne}^3} = \emptyset$, i.e., no access point is shared among the three entities. Essentially, this means that the framework presented in this paper aims at modelling networks of entities with one-to-one communication links, similarly as it is the case in, e.g., CCS [53]. However, this should not be considered as a disadvantage since one-to-many (broadcast) communication can easily be modelled within the proposed framework using the multiset labelling feature of transitions. The same is not true of models which do not allow multiset labelling.

The proof of the associativity of $\|$ is straightforward except for showing that the transitions generated by $\mathsf{pne}^1\|(\mathsf{pne}^2\|\mathsf{pne}^3)$ and $(\mathsf{pne}^1\|\mathsf{pne}^2)\|\mathsf{pne}^3$ are the same and have the same annotations. This part of the proof follows from the following observation. Suppose that $\mathsf{pne}^i = \langle \Sigma^i, \mathsf{acc}^i\rangle$ for $i = 1, 2, 3$. Denote, for $i \neq j$, $\mathsf{Id}_{ij} = \mathsf{Id}_{\mathsf{pne}^i} \cap \mathsf{Id}_{\mathsf{pne}^j}$, and $\mathsf{Id}_i = \mathsf{Id}_{ik} \cup \mathsf{Id}_{il}$, where $l, k \neq i$. Then one can show that the set of transitions of $\mathsf{pne}^1\|(\mathsf{pne}^2\|\mathsf{pne}^3)$ is given by

$$T \;=\; \bigcup_{i=1}^{3}\{t \in T_i \mid \bigoplus_{\mathsf{id}\in\mathsf{Id}_i} \mathsf{pne}^i(t) = \emptyset_{\mathsf{m}}\} \;\cup\; T_{12}^{\mathsf{new}} \;\cup\; T_{13}^{\mathsf{new}} \;\cup\; T_{23}^{\mathsf{new}} \;\cup\; T_{123}^{\mathsf{new}}$$

where the T^{new} sets are defined in the following way.

- Each T_{ij}^{new} comprises transitions t_{QR} such that $Q \in \mathsf{Ind}_{\Sigma^i}$ and $R \in \mathsf{Ind}_{\Sigma^j}$ are minimal sets of transitions satisfying, for all $\mathsf{id} \in \mathsf{Id}_{ij}$, $\mathsf{id}' \in \mathsf{Id}_i\setminus\mathsf{Id}_{ij}$ and $\mathsf{id}'' \in \mathsf{Id}_j\setminus\mathsf{Id}_{ij}$,

  $$\bigoplus_{t\in Q}\mathsf{pne}^i_{\mathsf{id}}(t) = \bigoplus_{t\in R}\mathsf{pne}^j_{\mathsf{id}}(t) \quad\text{and}\quad \bigoplus_{t\in Q}\mathsf{pne}^i_{\mathsf{id}'}(t) = \emptyset_{\mathsf{m}} = \bigoplus_{t\in R}\mathsf{pne}^j_{\mathsf{id}''}(t).$$

  Moreover, the connectivity and annotation of t_{QR} are inherited from Q and R.
- T_{123}^{new} comprises transitions $t_{Q_1Q_2Q_3}$ such that $Q_i \in \mathsf{Ind}_{\Sigma^i}$, for $i = 1, 2, 3$, are minimal sets of transitions satisfying, for all $i \neq j$ and $\mathsf{id} \in \mathsf{Id}_{ij}$,

  $$\bigoplus_{t\in Q_i}\mathsf{pne}^i_{\mathsf{id}}(t) = \bigoplus_{t\in Q_j}\mathsf{pne}^j_{\mathsf{id}}(t).$$

  Moreover, the connectivity and annotation of $t_{Q_1Q_2Q_3}$ are inherited from Q_1, Q_2 and Q_3.

One can see that the above indeed yields (up to net isomorphism) all the transitions of $\mathsf{pne}^1\|(\mathsf{pne}^2\|\mathsf{pne}^3)$. And, since all the formulae used are symmetric with respect to the three entities, it follows that $\mathsf{pne}^3\|(\mathsf{pne}^1\|\mathsf{pne}^2)$ is equal (up to net isomorphism) to $\mathsf{pne}^1\|(\mathsf{pne}^2\|\mathsf{pne}^3)$. Hence, since $\|$ is symmetric, we obtain that $\mathsf{pne}^1\|(\mathsf{pne}^2\|\mathsf{pne}^3) = (\mathsf{pne}^1\|\mathsf{pne}^2)\|\mathsf{pne}^3$.

We now turn to the properties of the bisimulation equivalence of entities. It is easy to see that $\approx$ is an equivalence relation. To start with, it is reflexive since $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}$, where bis is the identity relation on the set of markings reachable from the initial marking of pne. It is also symmetric since $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$ implies $\mathsf{pne}' \approx_{\mathsf{bis}^{-1}} \mathsf{pne}$. Finally, $\approx$ is transitive since $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$ and $\mathsf{pne}' \approx_{\mathsf{bis}'} \mathsf{pne}''$ implies $\mathsf{pne} \approx_{\mathsf{bis}\circ\mathsf{bis}'} \mathsf{pne}''$.

III. Proof of Theorem IV.1

Assume that $\mathsf{pne} = \langle \Sigma, \mathsf{acc} \rangle$ and $\mathsf{pne}' = \langle \Sigma', \mathsf{acc}' \rangle$ are entities such that $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$. Moreover, let $H \subseteq \mathsf{acc}$ and $H' \subseteq \mathsf{acc}'$ be sets of access points such that $\mathsf{Id}_H = \mathsf{Id}_{H'}$. Define $\mathsf{pne}^1 = \mathsf{hide}_H(\mathsf{pne}) = \langle \Sigma, \mathsf{acc}^1 \rangle$, $\mathsf{pne}^2 = \mathsf{hide}_{H'}(\mathsf{pne}') = \langle \Sigma', \mathsf{acc}^2 \rangle$, $\mathsf{Id} = \mathsf{Id}_{\mathsf{pne}} = \mathsf{Id}_{\mathsf{pne}'}$ and $\mathsf{Id}' = \mathsf{Id}_{\mathsf{pne}^1} = \mathsf{Id}_{\mathsf{pne}^2}$. We will show that $\mathsf{pne}^1 \approx_{\mathsf{bis}} \mathsf{pne}^2$. Suppose that $(M_1, M_1') \in \mathsf{bis}$ and consider three cases.

Case 1: $M_1\ [U\rangle_{\mathsf{pne}^1} M_2$ and $\mathsf{acc}^1(U) \neq \emptyset_{\mathsf{m}}$.[6]
Then, clearly, $\mathsf{acc}(U) \neq \emptyset_{\mathsf{m}}$ and so, by $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$, there are U' and M_2' such that $(M_2, M_2') \in \mathsf{bis}$ and $M_1' \xRightarrow{U'}_{\mathsf{pne}'} M_2'$ and, for all $\mathsf{id} \in \mathsf{Id}$, $\mathsf{pne}_{\mathsf{id}}(U) = \mathsf{pne}'_{\mathsf{id}}(U')$. Hence, clearly, $M_1' \xRightarrow{U'}_{\mathsf{pne}^2} M_2'$ and, since $\mathsf{Id}' \subseteq \mathsf{Id}$, for all $\mathsf{id} \in \mathsf{Id}'$, $\mathsf{pne}^1_{\mathsf{id}}(U) = \mathsf{pne}^2_{\mathsf{id}}(U')$.
Case 2: $M_1\ [U\rangle_{\mathsf{pne}^1} M_2$ where $\mathsf{acc}^1(U) = \emptyset_{\mathsf{m}}$ and $\mathsf{acc}(U) = \emptyset_{\mathsf{m}}$.
Then, by $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$, there is M_2' such that $(M_2, M_2') \in \mathsf{bis}$ and $M_1' \Longrightarrow_{\mathsf{pne}'} M_2'$. Clearly, the latter implies $M_1' \Longrightarrow_{\mathsf{pne}^2} M_2'$.
Case 3: $M_1\ [U\rangle_{\mathsf{pne}_1} M_2$ where $\mathsf{acc}^1(U) = \emptyset_{\mathsf{m}}$ and $\mathsf{acc}(U) \neq \emptyset_{\mathsf{m}}$.
Then, by $\mathsf{pne} \approx_{\mathsf{bis}} \mathsf{pne}'$, there are U' and M_2' such that $(M_2, M_2') \in \mathsf{bis}$ and $M_1' \xRightarrow{U'}_{\mathsf{pne}'} M_2'$ and, for all $\mathsf{id} \in \mathsf{Id}$, $\mathsf{pne}_{\mathsf{id}}(U) = \mathsf{pne}'_{\mathsf{id}}(U')$. Hence $M_1' \Longrightarrow_{\mathsf{pne}^2} M_2'$.

Note also that from Cases 2 and 3 it follows that if $M_1 \Longrightarrow_{\mathsf{pne}^1} M_2$ then there is M_2' such that $(M_2, M_2') \in \mathsf{bis}$ and $M_1' \Longrightarrow_{\mathsf{pne}^2} M_2'$. Since the above argument is symmetric with respect to pne^1 and pne^2, it follows that $\mathsf{pne}^1 \approx_{\mathsf{bis}} \mathsf{pne}^2$.

IV. Proof of Theorem IV.2

Since $\approx$ is commutative and associative, it suffices to show that if $\mathsf{pne}^i = \langle \Sigma^i, \mathsf{acc}^i \rangle$ $(i = 0, \ldots, 4)$ are entities such that $\mathsf{pne}^1 = \mathsf{pne}^0 \| \mathsf{pne}^3$, $\mathsf{pne}^2 = \mathsf{pne}^0 \| \mathsf{pne}^4$ and $\mathsf{pne}^3 \approx \mathsf{pne}^4$ then $\mathsf{pne}^1 \approx \mathsf{pne}^2$.

Assume that $\mathsf{pne}^3 \approx_{\mathsf{bis}} \mathsf{pne}^4$, $\mathsf{Id} = \mathsf{Id}_{\mathsf{pne}^0} \cap \mathsf{Id}_{\mathsf{pne}^3} = \mathsf{Id}_{\mathsf{pne}^0} \cap \mathsf{Id}_{\mathsf{pne}^4}$ and $\Sigma^i = \langle S^i, T^i, F^i, M_0^i \rangle$ for $i = 0, \ldots, 4$. We will show that

$$\begin{aligned} \mathsf{bis}' \;=\; \{(M \cup M_3, M \cup M_4) \;\mid\; & M \in [M_0^0\rangle \;\wedge\; (M_3, M_4) \in \mathsf{bis} \;\wedge \\ & M \cup M_3 \in [M_0^1\rangle \;\wedge\; M \cup M_4 \in [M_0^2\rangle\ \}. \end{aligned}$$

is a bisimulation relation such that $\mathsf{pne}^1 \approx_{\mathsf{bis}'} \mathsf{pne}^2$. Clearly, $(M_0^1, M_0^2) \in \mathsf{bis}'$.

[6] To avoid ambiguity we annotate the reachability relations with the names of relevant entities.

Suppose that $(M_1, M_2) = (M \cup M_3, M \cup M_4) \in \mathsf{bis}'$ and $M_1\ [U\rangle\ M_1'$. We may partition U as $U = U_0 \uplus U_3 \uplus U_{03}$ where $U_0 \subseteq T^0$, $U_3 \subseteq T^3$ and $U_{03} = \{t_{Q_iR_i} \mid i \in I\}$ for some index set I (see the definition of $\|$). From the definition of $t_{Q_iR_i}$ and the 1-safeness of all the nets involved in this proof, it follows that U_0 and Q_i (for $i \in I$) are mutually disjoint sets of transitions of Σ^0 and ${}^\bullet V_0 \subseteq M$, where $V_0 = U_0 \cup \bigcup_{i \in I} Q_i$. Similarly, U_3 and R_i (for $i \in I$) are mutually disjoint sets of transitions of Σ^3 and ${}^\bullet V_3 \subseteq M_3$, where $V_3 = U_3 \cup \bigcup_{i\in I} R_i$. Let M' and M_3' be markings such that $M\ [V_0\rangle\ M'$ and $M_3\ [V_3\rangle\ M_3'$. Clearly, $M_1' = M' \cup M_3'$.

Assume now that $U_{03} \neq \emptyset$ (the case when $U_{03} = \emptyset$ is similar, even slightly simpler). Then $V_0 \neq \emptyset \neq V_3$ and $\mathsf{acc}^0(V_0) \neq \emptyset_\mathsf{m} \neq \mathsf{acc}^3(V_3)$. From $\mathsf{pne}^3 \approx_\mathsf{bis} \mathsf{pne}^4$ and $(M_3, M_4) \in \mathsf{bis}$ it follows that there are V_4 and M_4' such that $(M_3', M_4') \in \mathsf{bis}$, $M_4 \overset{V_4}{\Longrightarrow} M_4'$ and, for all $\mathsf{id} \in \mathsf{Id}_{\mathsf{pne}^3}$, $\mathsf{acc}^3_\mathsf{id}(V_3) = \mathsf{acc}^4_\mathsf{id}(V_4)$. Let $V_4' = V_4 \cap T^2$ and $V_4'' = V_4 \setminus V_4'$. From the definition of $\|$ it follows that, for all $\mathsf{id} \in \mathsf{Id}$, $\mathsf{pne}^4_\mathsf{id}(V_4') = \emptyset_\mathsf{m}$ and $\mathsf{pne}^4_\mathsf{id}(V_4'') = \mathsf{pne}^0_\mathsf{id}(Q)$, where $Q = \bigcup_{i\in I} Q_i$. Hence there is a set of independent transitions $U_{04} \subseteq T^2$ such that ${}^\bullet U_{04} = {}^\bullet Q \cup {}^\bullet V_4''$, $U_{04}{}^\bullet = Q^\bullet \cup V_4''{}^\bullet$. Define $U' = U_0 \uplus V_4' \uplus U_{04}$ and $M_2' = M' \cup M_4'$. We have $M_2 \overset{U'}{\Longrightarrow} M_2'$, $(M_1', M_2') \in \mathsf{bis}'$ and, for every $\mathsf{id} \in \mathsf{Id}_{\mathsf{pne}^1}$, $\mathsf{pne}^1_\mathsf{id}(U) = \mathsf{pne}^2_\mathsf{id}(U')$.

Acknowledgments

The research reported in this paper has been supported by the Russian Fund for Basic Research, Grant 96-01-00177, the Royal Society Grant 638053.P466, and the NATO Grant SA.12-5-02(CN.NIG 960067)454/JPN/098.

References

[1] N.A.Anisimov, "A Notion of Petri Net Entity for Communication Protocol Design", Report, Institute for Automation and Control Processes, The Russian Academy of Sciences, Vladivostok, 1989.

[2] N.A.Anisimov, "An Algebra of Regular Macronets for Formal Specification of Communication Protocols". *Computers and Artificial Intelligence*, vol. 10, 1991, pp.541–560.

[3] N.A.Anisimov, "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, 1991, pp.440–450.

[4] N.A.Anisimov, A.A.Kovalenko, P.A.Postupalski, "Compositional Petri Net Environment", in: *Proc. of the 1994 IEEE Symposium on Emerging Technologies & Factory Automation: ETFA'94,* Tokyo, Japan, Nov. 1994, pp.420–427.

[5] N.A.Anisimov, A.A.Kovalenko, "Towards Petri Net Calculi based on Synchronization via Places", in *Proc. of the 1995 IEEE Symposium on Parallel Algorithms/Architecture Synthesis*, Aizu, Japan, March 1995, pp.264–270.

[6] N.Anisimov, M.Koutny, "On Compositionality and Petri Nets in Protocol Engineering", in: *Protocol Specification, Testing and Verification XV*, P.Dembiǹiski, M.Šredniawa eds., Chapman & Hall, 1995, pp.71-86.

[7] P.Azema, J.M.Ayache, and B.Berthomieu, "Design and Verification of Communication Procedures, a Bottom–Up Approach", in *Proc. 3rd Conference on Software Eng.*, Atlanta, 1978, pp.168–174.

[8] M.Y.Baerman, M.C.Wilbur-Ham, J.Billington, "Specification and Analysis of the OSI Class 0 Transport Protocol", in *Proc. 7th Int. Conf. Comput. Commun.*, 1984, pp.597–602.

[9] J.C.M.Baeten and W.P.Weijland: *Process Algebra*, Cambridge Tracts in Theoretical Computer Science, 1990.

[10] M.Barbeau, G.V.Bochmann: "A Subset of Lotos with the Computational Power of Place/Transition–Nets", in *Proc. 14th Int. Conf. on Application and Theory of Petri Nets*, ed. M.Ajmone Marsan, Chicago, IL, Springer Lecture Notes in Computer Science, vol. 691, 1993.

[11] B.Baumgarten, P.Ochsenschläger, R.Prinoth, "Building Blocks for Distributed System Design", in *Protocol Specification, Testing, and Verification, V: Proc. of the IFIP WG 6.1 5th International Workshop*, ed. M.Diaz, North–Holland, 1986, pp.19–38.

[12] L.Bernardinello and F.DeCindio, "A Survey of Basic Net Models and Modular Classes", in *Advances in Petri Nets 1992*, ed. G.Rozenberg, Springer Lecture Notes in Computer Science, vol. 609, pp.304–354, 1992.

[13] G.Berthelot and R.Terrat, "Modelling and Proofs of a Data Transfer Protocol by PREDICATE / TRANSITION Nets", *Informatik–Fachberichte,* vol. 52, Springer-Verlag,1982, pp.251–257.

[14] G.Berthelot and R.Terrat, "Petri Nets Theory for the Correctness of Protocols", *IEEE Trans. Commun.*, vol.30, 1982, pp.2497–2505.

[15] E. Best, R. Devillers, "Sequential and Concurrent Behaviour in Petri Net Theory", *Theoretical Computer Science,* vol. 55, no.1, 1988, pp.87–136.

[16] E.Best, R.Devillers, J.G.Hall, "The Box Calculus: a New Causal Algebra with Multi–label Communication", in *Advances in Petri Nets 1992*, Springer Lecture Notes in Computer Science vol. 609, 1992, pp.21–69.

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

[18] E. Best, M. Koutny, " A Refined View of the Box Algebra", in: *Proc. Petri Net Conference'95*, G. De Michelis, M. Diaz (eds). Springer-Verlag, Lecture Notes in Computer Science Vol. 935, pp.1–20, 1995.

[19] E.Best and H.G.Linde-Göers, "Compositional Process Semantics of Petri Boxes", in *Proc. of Mathematical Foundations of Programming Semantics,* Lecture Notes in Computer Science , Springer, 1993.

[20] J.Billington, "Abstarct Specification of the ISO Transport Service Definition Using Labelled Numerical Petri Nets", in: *Protocol Specification, Testing, and Verification, III: Proc. of the IFIP WG 6.1 3rd International Workshop,* ed. H.Rudin and C.H.West, North–Holland, 1983, pp.173–185.

[21] J.Billington, G.H.Wheeler, and H.C.Wilbur–Ham, " PROTEAN: A High–level Petri Net Tool for the Specification and Verification of Communication Protocols". *IEEE Trans. Soft. Eng.*, vol.14, no.3, pp.301–315, Mar. 1988.

[22] G.V.Bochmann, C.A.Sunshine. "Formal Methods in Communication Protocol Design". *IEEE Trans. Commun.*, col.COM–28, pp.624–631, 1980.

[23] T.Bolognessi, E.Brinksma: "Introduction to the ISO Specification Language LOTOS". *Computer Network and ISDN Systems*, vol.14, pp.25–29, 1987.

[24] G.Boudol and I.Castellani: "Flow Models of Distributed Computations: Event Structures and Nets", *Rapport de Recherche*, INRIA, Sophia Antipolis, 1991.

[25] W.Brauer, W.Reisig, G.Rozenberg (eds). "Petri Nets". Part I and II, in: Proc. of an Advanced Course, Bad Honnef. *Lecture Notes in Computer Science,* vol.254/255, Springer–Verlag, 1987.

[26] S.Budkowski, P.Dembinski: "An Introduction to Estelle: A Specification Language for Distributed Systems". *Computer Network and ISDN Systems,* vol.14, pp.3–23, 1987.

[27] H.J.Burkhardt, H.Eckert, and R.Prinoth, "Modelling of OSI–Communication Services and Protocols Using Predicate/Transition Nets", in: *Protocol Specification, Testing, and Verification,IV: Proc. of the IFIP WG 6.1 4th International Workshop,* ed.Y.Yemini, R.Strom, and S.Yemini, North–Holland, 1984, pp.165–192.

[28] W.Cellary, M.Saikowski, M.Stroiński, "Defining a Transport Layer Using Numerical Petri Nets", in: *First Int. Conf. Comut. and Appl.,* Beijing, 20–22 June, 1984, pp.353–360.

[29] T.Y.Choi, R.E.Miller, "Protocol Analysis and Synthesis by Structured Partitions", *Computer Network and ISDN Systems,* vol.11, pp.364–381, 1987.

[30] C.–H.Chow, M.G.Gouda, S.S.Lam, "A Discipline for Constructing Multiphase Communication Protocols", *ACM Transactions on Computer Systems,* vol.3, pp.315–343, 1985.

[31] J.P.Courtriat, J.M.Ayache, and B.Algayers, "Petri Nets are Good for Protocols", *ACM Computer Communication Review,* vol.14, pp.66–74, 1984.

[32] A.Danthine, Petri Nets for Protocol Modelling and Verification, in: *Proc. Computer Networks and Teleprocessing Symp.*, Budapest, 1977, pp.663–685.

[33] M.Devy and M.Diaz, "Multilevel Specification and Validation of the control in Communication Systems", in: *First Int.Conf.on Distrib. Comput. Syst.,* 1979, pp.43–50.

[34] M.Diaz: " Modeling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models". em Computer Networks, vol.6, pp.419–441, 1982.

[35] M.Diaz: "Petri Net Based Models in the Specification and Verification of Protocols". em Lecture Notes in Computer Science, vol.255, Springer, pp.135–170, 1987.

[36] *The Formal Description Technique Estelle: Results of the ESPRIT/SEDOS Project*, ed. M.Diaz, et al, North–Holland, 1989.

[37] *The Formal Description Technique LOTOS: Results of the ESPRIT/SEDOS Project*, ed. P.H.J.Van Eijk, et al, North–Holland, 1989.

[38] H.G.Genrich. "Predicate/Transition Nets". *Lecture Notes in Computer Science,* vol.254, Springer–Verlag, pp.207–247, 1987.

[39] Godefroid P. and Wolper P.: "Partial-order Methods for Temporal Verification", in: *Proc. Concur'93,* Lecture Notes in Computer Science, vol.715, Springer, pp.233–246, 1993.

[40] C.A.R. Hoare, "Communicating Sequential Processes", Prentice Hall, 1985.

[41] G.Holzmann, *Design and Validation of Computer Protocols,* Printice–Hall, 1991.

[42] ISO, IS 7498, Information Processing Systems – Open System Interconnections – Basic reference Model, 1983.

[43] K.Jensen, "Coloured Petri Nets", *Lecture Notes in Computer Science,* vol.254, Springer–Veralg, pp.248–299, 1987.

[44] P.C.Kanellakis, S.C.Smolka, "CCS Expressions, Finite State Processes and Three Problems of Equivalence", in: *Proc. of the Second ACM Symposium on Principles of Distributed Computing,* 1983.

[45] V.E.Kotov: "An Algebra for Parallelism Based on Petri Nets". *Lecture Notes in Computer Science,* vol.64, Springer, pp.39–55, 1978.

[46] M.Koutny: "Partial Order Semantics of Box Expressions". in: *15th International Conference on Application and Theory of Petri Nets*, Lecture Notes in Computer Science, Springer, 1994.

[47] M.Koutny and E.Best, "Operational and Denotational Semantics for the Box Algebra". *Theoretical Computer Science*. To appear, 1997.

[48] S.S.Lam, A.U.Shankar. "Protocol Verification via Projections". *IEEE Trans. Software Eng.,* vol.SE-10, no.4, pp.325-342, Jul.1984.

[49] M.T.Liu, "Protocol Engineering", *Advances in Computers,* vol.29, pp.80-195, 1989.

[50] P.M.Merlin: " Methodology for The Design and Implementation of Communication Protocols". *IEEE Trans. Commun.* vol.COM-24, pp.614–621, 1976.

[51] P.M.Merlin, D.J.Farber. "Recoverability of Communication Protocols — Implication of a Theoretical Study". *IEEE Trans. Commun.,* vol.COM-24, pp.1036-1043, 1976.

[52] P.M.Merlin, "Specification and Validation of Protocols", *IEEE Trans. Commun.,* vol. COM-27, pp.1671-1680, 1979.

[53] R.Milner: *Communication and Concurrency.* Prentice Hall (1989).

[54] K.Muller, "Constructable Petri Nets". *Electron. Int. Verrarb. Kybern. EIK,* vol.21, pp.171-199, 1985.

[55] T.Murata, "Petri Nets", *Proc. IEEE,* vol.77, pp.541–580, 1989.

[56] J.D.Noe, G.J.Nutt, "Macro E–nets for Presentation of Parallel Systems", *IEEE Trans. Comput.,* vol.22, pp.718-727, 1973.

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

[58] D.Peled: "Combining Partial Order Reductions with On-The-Fly Model Checking", in: *Proc. 6th International Conference on Computer Aided Verification*, Stanford, CA, Lecture Notes in Computer Science, vol.818 ,1994, pp.377-390.

[59] J.L.Peterson. *Petri Net Theory and the Modelling of Systems* Prentice–Hall Inc., 1981.

[60] A.F.Petrenko, "On the Specification and Verification of Protocols Using Petri Nets", in: *Proc. 5th Int. Conf. Comput. Commun.*, Atlanta, 1980, pp.385–390.

[61] T.F.Piatkowski, "An Engineering Discipline for Distrubuted Protocol Systems", in: *Proc. IFIP Workshop on Protocol Testing — Towards Proof?* 1981, pp.177–215.

[62] W. Reisig: *Petri Nets. An Introduction.* Springer-Verlag, EATCS Monographs on Theoretical Computer Science Vol. 3, 1985.

[63] H.Rudin: " Network Protocols and Tools to Help Produce Them". In: *Ann. Rev. Comput. Sci.,* vol.2, pp.291–316, 1987.

[64] R.Saraco, P.A.J.Tilanus, "CCITT SDL: Overview of the Language and its Applications", *Computer Networks and ISDN Systems,* vol.13, pp.65–74, 1987.

[65] Special Issue on Protocol Engineering, ed. M.T.Lee, *IEEE Trans. Computers,* vol.40, no.4, Apr. 1991.

[66] C.Sunshine. "A Formal Techniques for Protocol Specification and Verification". *Computer,* vol.12, pp.20-27, 1979.

[67] I.Suzuki and T.Murata, "A Method for Stepwise Refinement and Abstraction of Petri Nets". *Journal of Computer and System Science,* vol.27, pp.51-76, 1979.

[68] I.Suzuki, "Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets", *IEEE Trans. Software Eng.*, vol.16, no.6, pp.1273-1281, Nov. 1990.

[69] F.J.W.Symons. "Modelling and Analysis of Communication Protocols using Numerical Petri Nets". Ph.D.Thesis, *Department of Electrical Engineering Science,* University of Essex, England, 1978.

[70] D.Taubner: “Finite Representation of CCS and TCSP Programs by Automata and Petri Nets”. *Lecture Notes in Computer Science,* vol 369, Springer, 1989.

[71] R.Valette, “Analysis of Petri Nets by Stepwise Refinements”. *Journal of Computer and System Science,* vol.18, pp.35-46, 1979.

[72] A. Valmari, “A Stubborn Attacck on State Explosion”. in: *Computer-Aided Verification, AMS-ACM DIMACS Series in Discrete mathematics and Theoretical Computer Science,* vol.3, American Mathematical Society, 1990, pp.25-41.

[73] C.A.Vissers, G.Scollo, M.Van Sinderen, “Architecture and Specification Style in Formal Description of Distributed Systems”, in: *Protocol Specification, Testing, and Verification, VIII: Proc.of the IFIP WG 6.1 8th International Symposium,* eds. S. Aggarval, K. Sabnani, North–Holland, 1988.

[74] K.Voss, “Using Predicate/Transition–Nets to Model and Analyze Distributed Database Systems”, *IEEE Trans. Soft. Eng.* vol.SE–6, pp.539-544, 1980.

[75] S.T.Vuong, D.D.Cowan: “A Decomposition Method for the Validation of Structured Protocols”. In: *Proc. Conf. INFOCOM'82*, pp.209-220, 1982.

[76] F.E.Wang, K.Gilda, and A.Rubinstein, “A Coloured Petri Net Model for Connection Management Services in MMS”, *Computer Communication Review,* vol.19, pp.76-98, 1989.

[77] H.Zimmermann, “On Protocol Engineering”, in: *Proc.IFIP Information Processing 83,* pp.283-292, 1983.

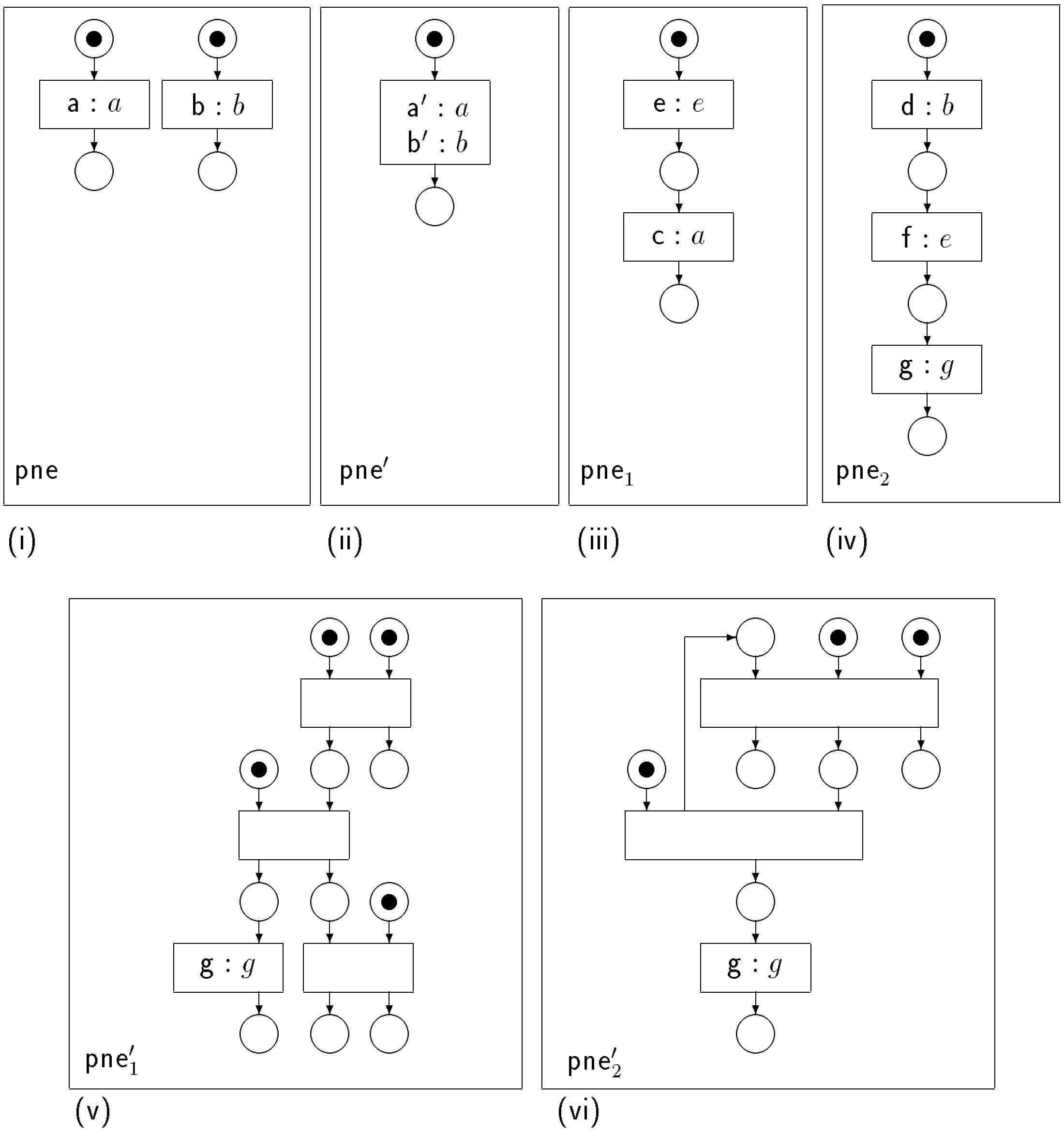

Fig. 13. Bisimulation must be the same for all access points, $\mathsf{Id_a} = \mathsf{Id_{a'}} = \mathsf{Id_c}$, $\mathsf{Id_e} = \mathsf{Id_f}$ and $\mathsf{Id_b} = \mathsf{Id_{b'}} = \mathsf{Id_d}$.