Universiteit
Leiden
The Netherlands

# Protocol Link Nets

## A Flexible Method for Composing Petri Nets

Dennis Roos

Supervisors:
Jetty Kleijn
Hendrik Jan Hoogeboom

12 August 2020

Master Thesis  Computer Science and Advanced Data Analytics

Leiden Institute of Advanced Computer Science (LIACS)

**Abstract.** Petri nets are a well-established framework for the modeling of distributed systems. Nowadays, there is an increasing demand for formal models for systems consisting of thousands of semi-independent parts, particularly in economics and biology. A natural way to create such models is by designing relatively small Petri nets for each part, and then composing them based on how the parts interact with each other. However, previous work on Petri net composition has mostly focused on fixed binary composition. In practice, this is insufficient for modeling the various complex ways a large number of components can interact. In this thesis we propose protocol link nets as a flexible method for combining Petri nets in different manners. We show that protocol link nets encompass existing methods of composition. We discuss a number of useful properties of the method, and present case studies where protocol link nets are useful for building a large model composed of many smaller models.

## 1 Introduction

Concurrent systems are all around us, from companies and financial institutions, to biological systems like ecology and gene expression, to theoretical applications in mathematics and computer science. To further our understanding of how these systems work, and how we can influence them, we must create models of them. A common and well-established method of modeling concurrent systems is through Petri nets [9] [12], which we will be using in this thesis.
Concurrent systems, almost by definition, rarely exist in isolation. In industries each company has its own internal processes, but the company also interacts with other companies. In computing and networking there may be clusters and internal networks that communicate over a shared bus. In biology we can create a model for a single cell, but that cell is but one in billions in a body. It interacts with its neighbors, with bloodstreams and nerves, and with pathogens and immune systems. And on a larger scale each organism interacts with other organisms.
Therefore it is natural to seek for methods that allow us to compose Petri nets into one big Petri net as part of our modeling effort. Petri net composition has been widely researched and written on [6] [5] [2] [11], but this research is overwhelmingly focused on simple, binary composition. Their cases involve just two nets being composed in limited ways. In practice this is insufficient for practical complex models.
As an example, Figure 1 shows a biological network model for the metabolic system of *Mycoplasma pneumoniae* [16].

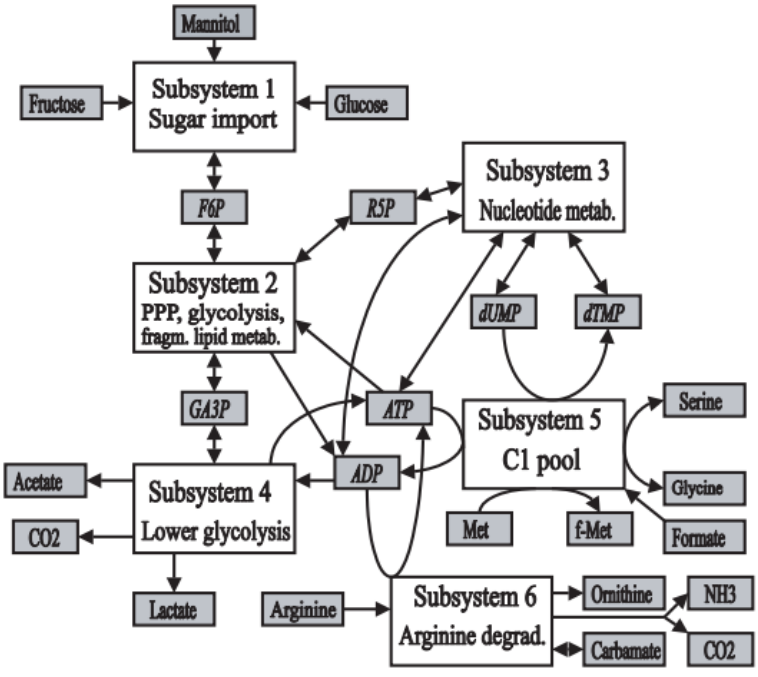


**Fig. 1.** The interactions in the metabolic system of a bacteria, from [16].

This is a system with many subsystems, dependencies, complicated different interactions. Simple uniform binary composition is not sufficient to model this. In addition to this complexity, in biology it is very common for us to learn more about the organisms we are modeling, changing our understanding and forcing us to change

or update the model to reflect this. There is currently no established method of Petri net composition which is suited for modeling such biological systems.

### 1.1 Our Proposal

In this thesis we present a method with a very different philosophy from those currently in use. One focused on versatility and allowing the user to easily implement, combine, and modify all kinds of composition strategies.
As mentioned before, we will be using Petri nets in this thesis. A Petri net is a graph where nodes are active or passive elements. Our proposal is as follows: Each element of a net receives a label called a *protocol label*. In addition, the user defines a set of *protocol nets*. Then, the user can repeatedly choose any set of elements from any number of nets to create a *link* between.
If one wishes to compose a group of Petri nets, the protocol net associated with the descriptions of selected elements is inserted 'between' them. Protocol nets can have any shape. For example, they can simply fuse the elements into one single element to create a synchronization, they can add an algorithm such as a semaphore between program nets, or describe interactions between biological pathways. Because users can define as many protocol nets as needed for the problem they're working on, and there are no limits to the form of any net involved, this is a versatile method that can easily implement and combine nearly any other composition algorithm.

We first recall some basic Petri net definitions in Chapter 2. In Chapter 3 we introduce our new variants of Petri nets: Descriptive Interface Nets, or DINs, and a special type of DIN, a protocol net. Chapter 4 shows how we use protocol nets to compose DINs. Next, we show that protocols can be used to create any net structure. Chapter 6 explores common Petri net composition methods from the literature, and shows how protocol nets can be used to simulate these methods. Chapter 7 has some case studies to show how protocol nets can be used in practical cases. Finally we compare our method with some other, non-Petri net methods of composition in Chapter 8.

We show that protocol nets are a new promising method of composition, and prove several properties they have. We explore how they relate to other methods of composition, and show that protocol nets are more flexible and practical than many other methods.

## 2 Preliminaries

The set $\mathbb{N}$ is the set of nonnegative integers $\{0, 1, 2, \ldots\}$. The set of positive integers is $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$.

For a function $f : A \to B$, we call $A$ the *domain* of $f$, denoted as $\mathrm{DOM}(f)$.

For two functions $f_1 : A_1 \to B_1$ and $f_2 : A_2 \to B_2$, with $A_1 \cap A_2 = \emptyset$, their union is $f_1 \cup f_2 : (A_1 \cup A_2) \to (B_1 \cup B_2)$, where $(f_1 \cup f_2)(a) = f_1(a)$ if $a \in A_1$, and $(f_1 \cup f_2)(a) = f_2(a)$ if $a \in A_2$.

For a function $f : A \to B$, and a subset $C \subseteq A$, the *restriction of $f$ to $C$* is the function $f_{|C} : C \to B$, defined by $f_{|C}(a) = f(a)$, for all $a \in C$.

For two sets $A$ and $B$, we denote their *disjoint union* with $A \uplus B$, i.e. the union of $A$ and $B$ under the assumption that they are disjoint.

Given a finite set $A$, we use $|A|$ to denote its cardinality.

Given a set $A$, the set $A^* = \bigcup_{n=0}^{\infty} \{(a_1, \ldots, a_n) | a_1, \ldots, a_n \in A\}$.
It consists of all finite sequences of elements of $A$. We write $()$ for the empty sequence.

Given a sequence $X = (x_1, \ldots, x_n)$, we denote the elements of $X$ as $elts(X) = \{x_1, \ldots, x_n\}$. Given two sequences $X = (x_1, \ldots, x_n)$, $Y = (y_1, \ldots, y_m) \in A^*$, the *concatenation* of $X$ and $Y$, denoted as $X \circ Y$, is the sequence $(x_1, \ldots, x_n, y_1, \ldots, y_m)$.

### 2.1 Petri Nets

**Definition 1.** *A* Petri net *is a tuple $N = (P, T, F)$ where $P$ and $T$ are finite disjoint sets and $F : (P \times T) \cup (T \times P) \to \mathbb{N}$ a function. The elements of $P$ are the* places *of $N$; the elements of $T$ are the* transitions *of $N$, and $F$ is the* flow function *of $N$.* ■

Given a Petri net $N = (P, T, F)$, we refer to the the places and transitions of $N$ collectively as the *elements* of $N$. Two elements $x$ and $y$ of $N$ are *of the same type* if $x, y \in P$ or $x, y \in T$, i.e. if both are places or both are transitions. Given an element $x$ of $N$, ${}^\bullet x = \{y | F(y, x) \geq 1\}$ is the *preset of* $x$, and $x^\bullet = \{y | F(x, y) \geq 1\}$ is the *postset of* $x$.

Two Petri nets $N_1 = (P_1, T_1, F_1)$, and $N_2 = (P_2, T_2, F_2)$ are said to be *disjoint* if $(P_1 \cup T_1) \cap (P_2 \cup T_2) = \emptyset$, i.e. if they have no elements in common.

**Definition 2.** *Let $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ be two disjoint Petri nets. The* union *of $N_1$ and $N_2$, denoted by $N_1 \cup N_2$, is the net $N_1 \cup N_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2)$.* ■

Note that $N_1 \cup N_2$ is well-defined: $F_1 \cup F_2$ exists because $N_1$ and $N_2$ are disjoint.

In general we are not interested in concrete elements but rather the structure of the nets. This is why we use isomorphisms to change the identity of underlying identities of elements without changing the structure.

**Definition 3.** *Let $N = (P, T, F)$, $N' = (P', T', F')$ be two Petri nets, and let $\varphi : (P \cup T) \to (P' \cup T')$ be a bijective function such that $P' = \varphi(P)$, $T' = \varphi(T)$, $F'(\varphi(x), \varphi(y)) = F(x, y)$ for all $x, y \in P \cup T$. Then $N'$ is a* $\varphi$-renaming *of $N$, denoted by $N \equiv_\varphi N'$.* ■

If $N$ and $N'$ are two Petri nets for which there exists a bijection $\varphi$ such that $N'$ is a $\varphi$-renaming of $N$, then $N$ and $N'$ are said to be *isomorphic*, denoted by $N \equiv N'$. We refer to $\varphi$ as an *isomorphism* of $N$. We may also write $N' = \varphi(N)$.

**Lemma 1.** *The relation $\equiv$ is reflexive, symmetric, and transitive.*

*Proof.* Let $N$ be an arbitrary Petri net. Then:
$N = \varphi(N)$ where $\varphi$ is the identity function. Hence $\equiv$ is reflexive.
Let $\varphi$ be an isomorphism such that $N' = \varphi(N)$. The inverse function $\varphi^{-1}$ is a bijection because $\varphi$ is bijective. Now, let for all elements $z$ of $N$, $z'$ be the element of $N'$ such that $\varphi(z) = z'$. Then $F(x,y) = F(\varphi^{-1}(x'), \varphi^{-1}(y')) = F'(\varphi(\varphi^{-1}(x'), \varphi(\varphi^{-1}(y')) = F'(x', y')$. Hence $\equiv$ is symmetric.
Let $N_1, N_2, N_3$ be three Petri nets and $\varphi_1, \varphi_2$ be two isomorphisms such that $N_2 = \varphi_1(N_1)$ and $N_3 = \varphi_2(N_2)$. Then $F_1(x,y) = F_2(\varphi_1(x), \varphi_1(y)) =$
$F_3(\varphi_2(\varphi_1(x)), \varphi_2(\varphi_1(y)))$ and $\varphi_2 \circ \varphi_1$ is a bijection. Hence $N_3 = \varphi_2(\varphi_1(N_1))$ and so $\equiv$ is transitive. ■

Thus $\equiv$ forms an equivalence relation, partitioning the class of all Petri nets into isomorphism classes of nets that are renamings of each other.

**Definition 4.** *Let $N = (P, T, F)$ be a Petri net. A* marking *$M$ of $N$ is a function $M : P \to \mathbb{N}$.* ■

A marking assigns to each place a number. This number $M(p)$ indicates the local distributed state of a place $p$.

**Definition 5.** *A* Place/Transition system*, or* P/T system *for short, is a tuple $\mathcal{P} = (P, T, F, M_{in})$ where $(P, T, F)$ is a Petri net, called its* underlying net*, and $M_{in}$ is a marking of $(P, T, F)$.* ■

The behavior of a P/T system depends on what *firing rule* it uses. A firing rule indicates in which markings a transition can occur, and how the transition's occurrence changes the states of the places around it. We now present the typical firing rule.

**Definition 6.** *Let $N = (P, T, F)$ be a Petri net, and $M$ a marking of $N$, and let $t$ be a transition of $N$. Then $t$ is* enabled in M *if for all $p \in P$, $M(p) \geq F(p,t)$.*
*If $t$ is enabled, then it may* occur *in $M$, leading to the marking $M'$, defined by $M'(p) = (M(p) - F(p,t)) + F(t,p)$.* ■

Given a marking $M$ and a transition $t$, we write $M \xrightarrow{t}$ to denote that $t$ is enabled in $M$. Similarly we write $M \xrightarrow{t} M'$ to denote that the occurrence $t$ in $M$ leads to $M'$.

### 2.2 Graphical notation

For the graphical representation of Petri nets, we use the following conventions: Places are drawn as circles, transitions as rectangles, and the flow function is depicted as arcs between elements. An arc from element $x$ to $y$ indicates that $F(x,y) \geq 1$. If $F(x,y) = 0$ no arc is drawn between $x$ and $y$. A number next to an arc from $x$ to $y$ indicates $F(x,y)$. If no number is given, $F(x,y) = 1$. Markings are represented by drawing black dots (tokens): if $M(p) = k$ for a place $p$ and marking $M$, place $p$ contains $k$ tokens.

*Example 1.* Figure 2 shows a P/T system $\mathcal{P} = (P, T, F, M_{in})$. In this example, $F(p_5, t_3) = F(t_4, p_5) = 2$. $F(x,y) = 1$ for all other drawn arcs, and $F(x,y) = 0$ for all unconnected pairs of places and transitions or transitions and places. In this case, $M_{in}(p_1) = 1, M_{in}(p_5) = 2$, and $M_{in}(p) = 0$ for all other $p \in P$.

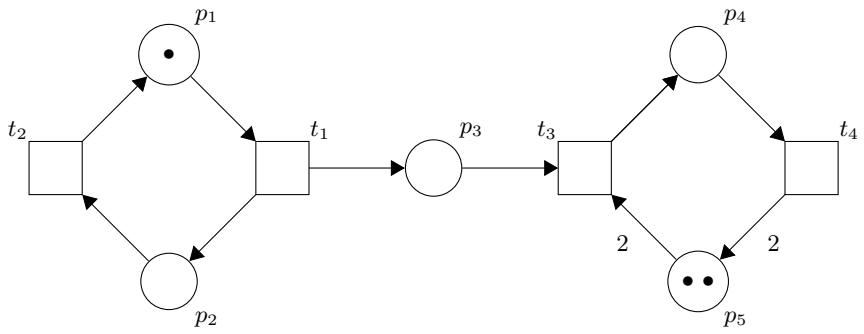


**Fig. 2.** $(P, T, F)$ with marking $M_{in}$.

In the initial marking $M_{in}$, only the transition $t_1$ is enabled. Firing it leads to a new marking $M_1$ depicted in Figure 3. In this marking both $t_2$ and $t_3$ are enabled.

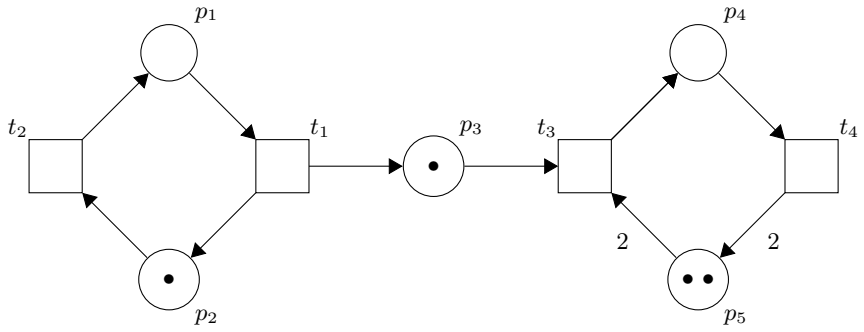


**Fig. 3.** $(P, T, F)$ with the marking $M_1$

The P/T System in Figure 2 is a variation of the well-known Producer-Consumer system. The left half of the system is a *producer*. Transitions $t_1$ and $t_2$ can occur alternatingly, with each loop producing a token that is deposited in the buffer place $p_3$. The right half of the system is a *consumer*. where transitions $t_3$ and $t_4$ can occur alternatingly as long as there are tokens in the buffer.

## 3 Protocols

In this section we present our new Petri net variations: Descriptive Interface Nets. In these nets, each element gets a *protocol label*, which will be used in the next sections.

### 3.1 Descriptive Interface Nets

Throughout this thesis, $\Pi$ is a fixed, possibly infinite set of *protocol labels*. Moreover, $\Pi = \Pi_P \biguplus \Pi_T$, with $\Pi_P$ a set of *place labels* and $\Pi_T$ a set of *transition labels*.

**Definition 7.** *A* descriptive interface net, DIN *for short, is a tuple* $\mathcal{D} = (P, T, F, \pi)$ *where* $(P, T, F)$ *is the* underlying net *of* $\mathcal{D}$*, and* $\pi : (P \cup T) \to \Pi$*, the* protocol labeling function *of* $D$*, is a function such that, for all places* $p \in P$*,* $\pi(p) \in \Pi_P$*, and for all transitions* $t \in T, \pi(t) \in \Pi_T$*.* ■

In graphical representations, the protocol label of an element is written inside the element, and its name is on the outside.

Two DINs are said to be disjoint if their underlying nets are disjoint.

**Definition 8.** *Let $D_1 = (P_1, T_1, F_1, \pi_1)$ and $D_2 = (P_2, T_2, F_2, \pi_2)$ be two disjoint DINs. Then their* union*, denoted by $D_1 \cup D_2$, is the DIN $D_1 \cup D_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, \pi_1 \cup \pi_2)$.* ■

$\pi_1 \cup \pi_2$ is well-defined since $D_1$ and $D_2$ are disjoint.

Using the basic properties of sets it follows immediately that the union of DINs as defined in Definition 8 is commutative and associative. Formally:

**Lemma 2.** *Let $D_1, D_2, D_3$ be pairwise disjoint DINs where $D_i = (P_i, T_i, F_i, \pi_i)$ for $i \in \{1, 2, 3\}$. Then the following statements hold:*
*1)* $D_1 \cup D_2 = D_2 \cup D_1$
*2)* $(D_1 \cup D_2) \cup D_3 = D_1 \cup (D_2 \cup D_3)$.

*Example 2.* Consider the DIN in Figure 4. It is the union of three disjoint DINs. Places $p_1$ and $p_2$ have protocol label $\alpha$, while transition $t_1$ is given protocol label $\beta$. The labels of all other elements are not depicted.

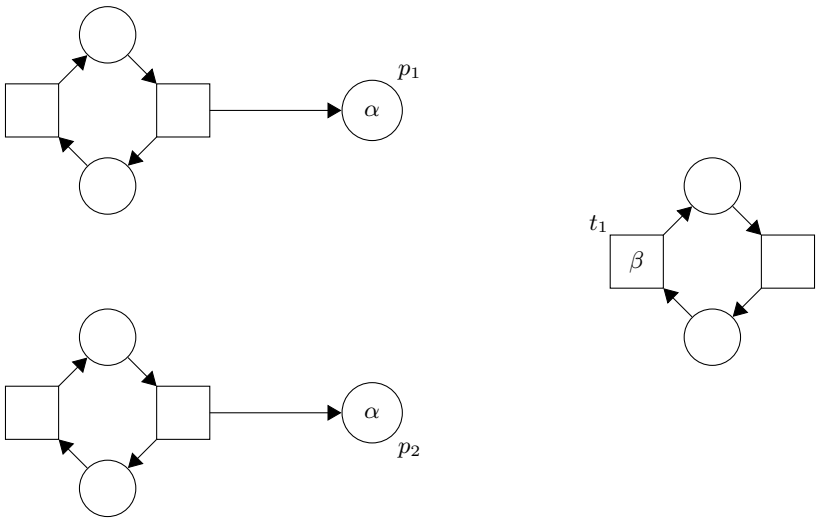


**Fig. 4.** A descriptive interface net.

**Definition 9.** *Let $\mathcal{D} = (P, T, F, \pi)$, $\mathcal{D}' = (P', T', F', \pi')$ be two DINs, and let $\varphi : (P \cup T) \to (P' \cup T')$ be a bijective function such that $(P, T, F) \equiv_\varphi (P', T', F')$, and $\pi'(\varphi(x)) = \pi(x)$ for all $x \in (P \cup T)$. Then $\mathcal{D}'$ is a* $\varphi$-renaming *of $\mathcal{D}$, denoted by $\mathcal{D} \equiv_\varphi \mathcal{D}'$.* ■

If $\mathcal{D}$ and $\mathcal{D}'$ are two DINs for which there exists a bijection $\varphi$ such that $D'$ is a $\varphi$-renaming of $D$, then $D$ and $D'$ are said to be *isomorphic*, denoted by $\mathcal{D} \equiv \mathcal{D}'$. We refer to $\varphi$ as an *isomorphism* of $D$. We may also write $\mathcal{D}' = \varphi(\mathcal{D})$.

**Lemma 3.** *The relation $\equiv$ between DINs is reflexive, symmetric, and transitive.*

*Proof.* We refer to the proof of Lemma 1, which we only have to extend to include the protocol labeling functions. So, let $D, D', D_1, D_2, D_3$ be DINs with underlying Petri nets $N, N', N_1, N_2, N_3$ respectively, and protocol labeling functions $\pi, \pi', \pi_1, \pi_2, \pi_3$ respectively. Assume that $\varphi, \varphi_1$, and $\varphi_2$ are isomorphisms such that $D' = \varphi(D), D_2 = \varphi_1(D_1)$, and $D_3 = \varphi_2(D_2)$.
For reflexivity with $\hat{\varphi}$ the identity mapping, we have $\pi \circ \hat{\varphi} = \pi$, as required.
For symmetry, we use that $\pi' \circ \varphi = \pi$ and so $\pi \circ \varphi^{-1} = (\pi' \circ \varphi) \circ \varphi^{-1} = \pi'$ as required.
Finally, transitivity follows from the observation that $\pi_3 \circ (\varphi_2 \circ \varphi_1) = (\pi_3 \circ \varphi_2) \circ \varphi_1 = \pi_2 \circ \varphi_1 = \pi_1$. ■

### 3.2 Protocol Nets

We now define protocol link nets, variations of DINs with a subset of elements selected as *anchors*. These protocol link nets will be 'added' to the overall DIN when composing through the anchor elements, as shown in section 4.

**Definition 10.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN, and $n \in \mathbb{N}^+$. An* anchor function (of length $n$) *of $\mathcal{D}$ is a function $A : \{1, 2, \ldots, n\} \to P \cup T$.*

**Definition 11.** *Let $n \in \mathbb{N}^+$. A* $n$-sided protocol link net*, or simply* protocol net*, is a tuple $\mathcal{L} = (P, T, F, \pi, A)$ where $(P, T, F, \pi)$ is a descriptive interface net, called the* underlying DIN *of $\mathcal{L}$, and $A : \{1, 2, \ldots, n\} \to P \cup T$ an* anchor function *of length $n$ of $(P, T, F, \pi)$.* ■

We use $PROT$ to denote the class of protocol link nets.

An anchor function $A$ of length $n$ of a protocol link net $\mathcal{L}$ defines a sequence $(A(1), \ldots, A(n))$ of elements of $\mathcal{L}$. Each $A(i)$ is called an *anchor*. Given an anchor $x$ of $\mathcal{L}$, we define its *anchor numbers* as the set $A^{-1}(x)$. In graphical notation, anchors are drawn shaded grey, and the anchor numbers of an element are written in square brackets below its name. See Figure 5 for an example.

We extend the definitions of renaming and isomorphism to protocol nets:

**Definition 12.** *Let $n \in \mathbb{N}^+$. Let $\mathcal{L} = (P, T, F, \pi, A)$ and $\mathcal{L}' = (P', T', F', \pi', A')$ be $n$-sided protocol link nets, and let $\varphi$ be an isomorphism such that $(P', T', F', \pi')$ is a $\varphi$-renaming of $(P, T, F, \pi)$, and $A'(i) = \varphi(A(i))$ for all $i \in \{1, \ldots, n\}$. Then $\mathcal{L}'$ is a* $\varphi$-renaming *of $\mathcal{L}$, denoted by $\mathcal{L} \equiv_\varphi \mathcal{L}'$.* ■

If $\mathcal{L}$ and $\mathcal{L}'$ are two protocol nets for which there exists a bijection $\varphi$ such that $\mathcal{L}'$ is a $\varphi$-renaming of $\mathcal{L}$, then $\mathcal{L}$ and $\mathcal{L}'$ are said to be *isomorphic*, denoted by $\mathcal{L} \equiv \mathcal{L}'$. We refer to $\varphi$ as an *isomorphism* of $\mathcal{L}$. We may also write $\mathcal{L}' = \varphi(\mathcal{L})$.

**Lemma 4.** *The relation $\equiv$ between protocol nets is reflexive, symmetric, and transitive.*

*Proof.* We refer to the proof of Lemma 3, which we only have to extend to include the anchor functions.
For reflexivity with $\varphi$ the identity mapping, we have $\varphi \circ A = A$, as required.
For symmetry, we use that $\varphi \circ A' = A$ and so $\varphi^{-1} \circ A = (\varphi \circ A') \circ \varphi^{-1} = A'$ as required.
Finally, transitivity follows from the observation that $A_3 = \varphi_2 \circ A_2 = \varphi_2 \circ (\varphi_1 \circ A_1) = (\varphi_2 \circ \varphi_1) \circ A_1$. ■

*Example 3.* In Figure 5 we see a protocol net $\mathcal{L}$. Its anchors are, in order, $p_1, p_2$, and $t_2$.

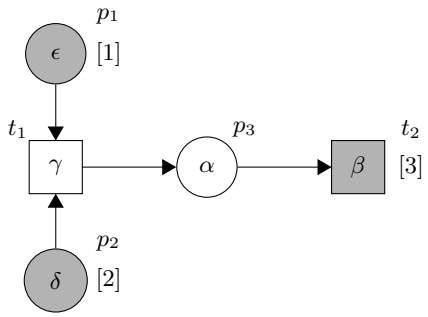


**Fig. 5.** A 3-sided protocol net. The gray elements are its anchors.

■

**Definition 13.** *Let $\Pi$ be an alphabet of protocol labels. Then a* protocol assignment *function is a function $\mathbb{L} : \Pi^* \to PROT$.*
*Let $n \in \mathbb{N}^+$ and let $\sigma = (\alpha_1, \ldots, \alpha_n) \in \Pi^*$ be a sequence of length $n$ for some $n \geq 1$. Then $\mathbb{L}((\alpha_1, \ldots, \alpha_n))$ is the $n$-sided protocol net associated with $\sigma$.* ■

Throughout this thesis we assume that $\mathbb{L}$ is a fixed protocol assignment function. We use a function like this (rather than just letting the user pick any protocol net they want for the given elements) to add some structure and allow the process to be automated and streamlined in the future. Both the protocol assignment function $\mathbb{L}$ and the set of labels $\Pi$ will be defined according to the user's needs. In this thesis we provide a framework in which users can easily define, add, combine, and modify their own sets of protocols to use. For example, a biologist may want to create protocol nets modeling biological pathways or biochemical reactions, while an economist could create protocols for businesses and other organizations interacting in various ways.

It should be made clear that the protocol assignment function will not always be completely specified, i.e. for all sequences in $\Pi^*$. In practice, only a few protocols may be used, and then only specific tuples of labels will be selected for composition. In the examples in the rest of this thesis we will only define the protocol nets that are relevant for the specific case discussed.

## 4 Protocol Composition

In this section we show how protocol nets are used to connect DINs. We first define the *fusing* operation, which enabled us to merge specific elements: If an element $x$ is fused to an element $y$ of the same type, then all flow connected to $x$ is transferred to $y$, and $x$ is deleted.

**Definition 14.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN. Let $n \in \mathbb{N}^+$. Let $\rho = (x_1, \ldots, x_n)$, and $\sigma = (y_1, \ldots, y_n)$ be distinct elements of $P \cup T$ such that all $x_1, \ldots x_n$ are distinct and $x_k$ and $y_k$ are of the same type for all $k \in \{1, \ldots, n\}$, and $elts(\rho) \cup elts(\sigma) = \emptyset$. The* fusing *of $\rho$ into $\sigma$ in $\mathcal{D}$, denoted by $\mathcal{D}[\rho \triangleright \sigma]$, is the DIN $(P', T', F', \pi')$ where $P' = P \setminus X$, $T' = T \setminus X$, $\pi' = \pi_{|(P' \cup T')}$, and for all $(a, b) \in (P' \times T') \cup (T' \times P')$: $F'(a, b) = F(a, b) + (\sum_{\{i \in \{1,\ldots,n | y_i = b\}\}} F(a, x_i)) + (\sum_{\{j \in \{1,\ldots,n | y_j = a\}\}} F(x_j, b)) + (\sum_{\{i,j \in \{1,\ldots,n | y_i = a \wedge y_j = b\}\}} F(x_i, x_j))$.*

It is important to note that the $y_i$ do not have to be distinct. If $y_i = y_j$, then $x_i$ and $x_j$ are both fused to $y_i$. The fused element will have the protocol label of $y_i$.

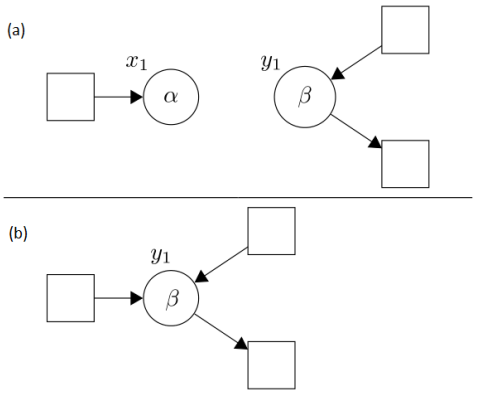


**Fig. 6.** A DIN before (a) and after (b) fusing $(x_1)$ onto $(y_1)$.

*Example 4.* Let $\mathcal{D}$ be the DIN in Figure 6(a). The DIN $\mathcal{D}[(x_1) \triangleright (y_1)]$ is depicted in Figure 6(b). The arcs that were connected to $x_1$ before are now connected to $y_1$.

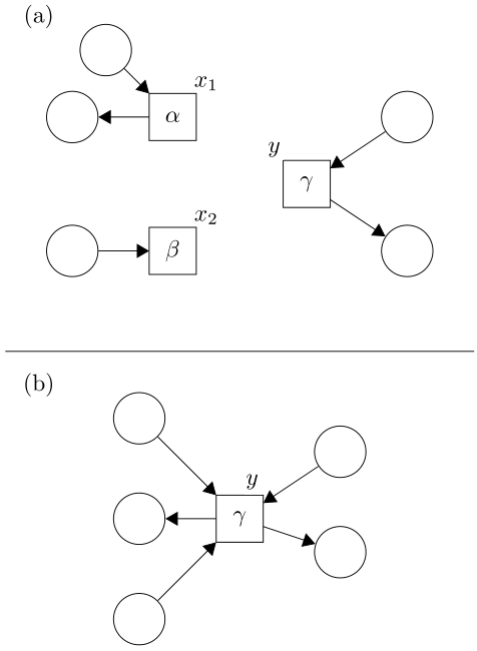


**Fig. 7.** A DIN before (a) and after (b) fusing $(x_1, x_2)$ onto $(y, y)$.

Let $\mathcal{D}'$ be the DIN in Figure 7(a), with $y_1 = y_2$. The DIN $\mathcal{D}'[(x_1, x_2) \triangleright (y, y)]$ is depicted in Figure 7(b). As $x_1$ and $x_2$ are mapped to the same element, they are merged together. ■

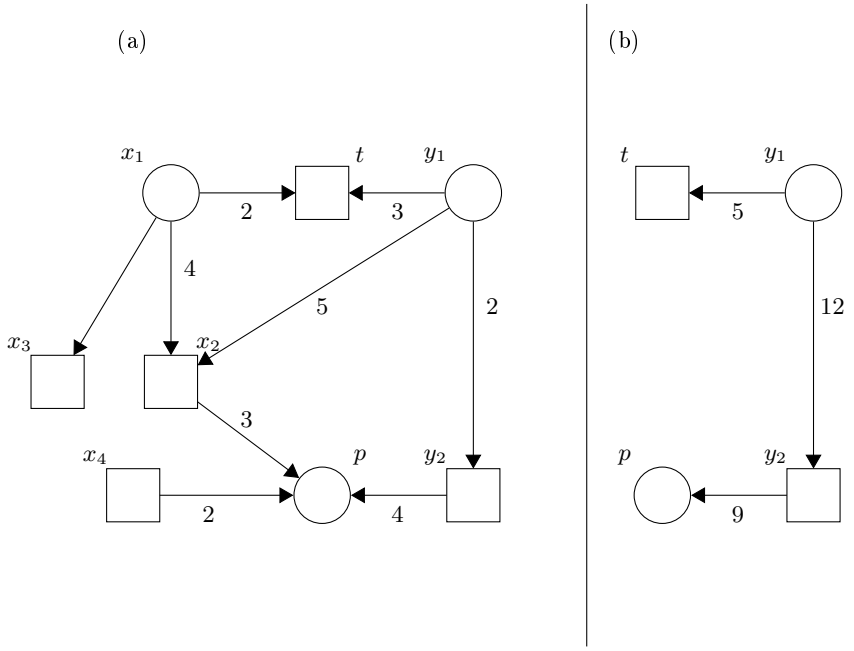


**Fig. 8.** A DIN before (a) and after (b) fusing $(x_1, x_2, x_3, x_4)$ onto $(y_1, y_2, y_2, y_2)$.

*Example 5.* Let $\mathcal{D}$ be the DIN depicted in Figure 8(a). Figure 8(b) shows the DIN $\mathcal{D}' = \mathcal{D}[(x_1, x_2, x_3, x_4) \triangleright (y_1, y_2, y_2, y_2)]$ (three of the $x_i$ are fused with the same element $y_2$). The flow from merged elements is added together:
$F'(y_1, t) = F(y_1, t) + F(x_1, t)$,
$F'(y_2, p) = F(y_2, p) + F(x_2, p) + F(x_3, p) + F(x_4, p)$ (where $F(x_3, p) = 0$),
$F'(y_1, y_2) = F(y_1, y_2) + F(y_1, x_2) + F(x_1, x_2) + F(x_1, x_3)$. ■

We now define how to *link* elements of a descriptive interface net using fusing and protocol nets. Note that this operations works on one DIN, not multiple distinct components. So when composing different DINs, we must consider them as a single DIN (using the DIN union from Definition 9). This approach allows us to combine any number of nets instead of a strict binary operation. It and also allows for adding links within a connected net, so if two nets have multiple distinct interactions (such as passing various data to each other) those connections can be represented with multiple protocol nets.

In what follows, a DIN $D = (P, T, F, \pi)$ and a protocol net $\mathcal{L} = (P', T', F', \pi', A)$ are said to be *disjoint* if $(P, T, F)$ and $(P', T', F')$ are disjoint Petri nets. Furthermore, if $D$ and $\mathcal{L}$ are disjoint, then their *union* is the DIN $D \cup \mathcal{L} = (P \cup P', T \cup T', F \cup F', \pi \cup \pi')$.

**Definition 15.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN. Let $n \in \mathbb{N}^+$. Let $(x_1, \ldots, x_n) \in (P \cup T)^*$ be a sequence of distinct elements of $D$. Let $\mathcal{L} = (P_L, T_L, F_L, \pi_L, A) = \mathbb{L}(\pi(x_1), \ldots, \pi(x_n))$ be the designated protocol net for the protocol labels of the $x_i$. Let $\varphi$ be an isomorphism of $\mathcal{L}$ such that $\mathcal{D}$ and $\varphi(\mathcal{L})$ are disjoint.*
*Then the* link in $\mathcal{D}$ for $(x_1, \ldots, x_n)$ and $\varphi$ (w.r.t. $\mathbb{L}$), $\lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi)$, *is the DIN* $(\mathcal{D} \cup \varphi(\mathcal{L}))[(x_1, \ldots, x_n) \triangleright (A(1), \ldots, A(n))]$. ■

In short, we link $n$ elements $x_1, \ldots, x_n$ using a protocol net $\mathcal{L}$ chosen based on their protocol labels and the protocol assignment function. Each $x_i$ is fused onto the corresponding anchor element $A(i)$ of $\mathcal{L}$. One important aspect to note here is

the role of the renaming function $\varphi$. If $\mathcal{L}$ and $\mathcal{D}$ are not disjoint, there will be a shared element $z$ that will have all incidences that it had in $\mathcal{D}$ and in $\mathcal{L}$. Thus we use a disjoint isomorphic copy of $\mathcal{L}$ to link the net assembled in $\mathcal{D}$.

Note that the elements $P' \cup T'$ of $\lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi)$ can be partitioned into two disjoint sets: $(P \cup T) \setminus \{x_1, \ldots, x_n\}$ and $\varphi(P_\mathcal{L}) \cup \varphi(T_\mathcal{L})$.

Because the elements (and thus the sets of arcs defined by the flows $F$ and $F_\mathcal{L}$) are disjoint, the flow of $(P', T', F', \pi') = \lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi)$ can be partitioned as follows:
First, the internal flow of $\mathcal{D}$: all pairs $(a, b)$ where $a, b \in (P \cup T) \setminus \{x_1, \ldots, x_n\}$ and $a, b$ not the same type. For these, $F'(a, b) = F(a, b)$.
Second, the internal flow of $\mathcal{L}$: all pairs $(a, b)$ where $\varphi^{-1}(a), \varphi^{-1}(b) \in (P_\mathcal{L} \cup T_\mathcal{L})$ and $a, b$ not both anchors and not of the same type. For these, $F'(a, b) = F_\mathcal{L}(\varphi^{-1}(a), \varphi^{-1}(b))$.
Third, the flow connecting $D$ and $\mathcal{L}$: all pairs $(a, b), (b, a)$ where $a, b$ not the same type, $b \in D$ and $\varphi^{-1}(a) = A(i)$ for some $i$. This is the flow between $x_i$ and $b$. For these $F'(a, b) = \sum_{\{i \in \{1,\ldots,n | A(i) = \varphi^{-1}(a)\}} F(x_i, b)$. and
$F'(b, a) = \sum_{\{i \in \{1,\ldots,n | A(i) = \varphi^{-1}(a)\}} F(b, x_i)$.
Finally, there is the flow between anchors: all pairs $(a, b)$ where $a, b$ not of the same type, $\varphi^{-1}(a) = A(i)$, and $\varphi^{-1}(b) = A(j)$ for some $i, j$. For these,
$F'(a, b) = F_\mathcal{L}(\varphi^{-1}(a), \varphi^{-1}(b)) + (\sum_{\{i,j \in \{1,\ldots,n | A(i) = \varphi^{-1}(a) \wedge A(j) = \varphi^{-1}(b)\}\}} F(x_i, x_j))$.
Note that, because $\mathcal{D}$ and $\mathcal{L}$ are disjoint, there was no flow between any $x_i$ and $A(j)$.

*Example 6.* We link the three components from Example 2 using the protocol net from Example 3. Let $\mathcal{D}$ be the DIN in Figure 4, and let $\mathcal{L}$ from Figure 5 be $\mathbb{L}(\alpha, \alpha, \beta)$. Let $\sigma = (p_1, p_2, t_1)$. We now create the DIN $\lambda(\mathcal{D}, \sigma, \varphi)$. Because $\mathcal{L}$ is the protocol net associated with the labels of $\sigma$, it is added to the DIN by fusing its anchors to $p_1, p_2$, and $t_1$. The protocol labels are overwritten by the labels of $\mathcal{L}$'s anchors because of the fusing. However, it changes nothing for $t_1$, as the protocol net also had $\beta$ as label for $A(3)$. Elements $t_1$ and $p_3$ in $\mathcal{L}$ are renamed with $\varphi$ to keep it disjoint from $\mathcal{D}$.

The combined net is now a modified producer-consumer, with two producers that both have to produce output for the consumer to receive a token.

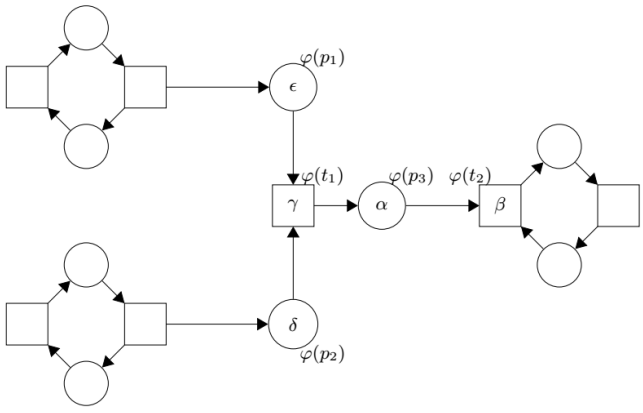


**Fig. 9.** The composed system.

### 4.1 Properties of Protocol Composition

In this section we will show some algebraic properties of protocol composition. First we show that which isomorphism for $\mathcal{L}$ we use does not matter for the structure of the linked net. Secondly, we show that when adding multiple links to distinct elements, we can add them in any order. Finally, we show that we can, under certain conditions, do multiple links at once with one protocol net.

**Lemma 5.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN, Let $n \in \mathbb{N}^+$. Let $(x_1, \ldots, x_n) \in (P \cup T)^*$ be a sequence of distinct elements of $\mathcal{D}$. Let $\mathcal{L} = \mathbb{L}(\pi(x_1), \ldots, \pi(x_n))$. Let $\varphi_1$ and $\varphi_2$ be two isomorphisms of $\mathcal{L}$ such that $\mathcal{D}$ is disjoint with $\varphi_1(\mathcal{L})$ and $\varphi_2(\mathcal{L})$. Then:*

$\lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi_1) \equiv \lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi_2)$.

*Proof.* Let $D_1 = (P_1, T_1, F_1, \pi_1) = \lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi_1)$ and $D_2 = (P_2, T_2, F_2, \pi_2) = \lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi_2)$. Let $\mathcal{L} = (P_L, T_L, F_L, \pi_L, A)$. Let $\varphi$ be the isomorphism of $\varphi_1(\mathcal{L})$ such that $\varphi(\varphi_1(\mathcal{L})) = \varphi_2(\mathcal{L})$. Let $\psi : (P_1 \cup T_1) \mapsto (P_2 \cup T_2)$ be a function such that $\psi(q) = q$ for $q \in (P \cup T)$, and $\psi(q) = \varphi(q)$ for $q \in (P_L \cup T_L)$.

We first show that $\psi$ is a bijection. Recall that linked nets can be partitioned into two sets of elements: elements from $D$ (with $X$ removed) and elements from $\mathcal{L}$. $\psi$ maps the elements from $D$ to themselves, and the elements from $\mathcal{L}$ are mapped bijectively because $\varphi$ is an isomorphism and thus a bijection over the elements of $\mathcal{L}$. We also note that because elements are mapped to themselves or their renamed counterparts, for all $q$, $\pi_2(q) = \pi_1(q)$ or $\pi_2(\varphi(q)) = \pi_1(q)$ depending on whether $q \in P \cup T$ or $q \in P_L \cup T_L$ respectively.

Finally we show that $F_1(q, r) = F_2(\psi(q), \psi(r))$ for all $q, r \in P_1 \cup T_1$. We discuss the different cases as outlined in the note after Definition 16:
$q, r \in (P \cup T) \setminus X$. Then, $\psi(q) = q$ and $\psi(r) = r$, so $F(q, r) = F_1(q, r) = F_2(q, r)$.
$q, r \in (\varphi_1(P_L) \cup \varphi_1(T_L)$. Then, $\psi(q) = \varphi_2(q)$ and $\psi(r) = \varphi_2(r)$, so $F_{\mathcal{L}}(\varphi_1^{-1}(q), \varphi_1^{-1}(r)) = F_1(q, r) = F_2(\psi(q), \psi(r))$.
$q \in (P \cup T), r \notin (P \cup T)$. This is an arc between the $D$ part and the $\mathcal{L}$ part. As $D$ and $\mathcal{L}$ were disjoint, this must be a result of fusing an element of $D$ with $\mathcal{L}$: $r$ was an anchor $A(i)$ of $\mathcal{L}$ that was fused with one or more of the $x_i$. So $F_1(q, r) = \sum_{\{i \in \{1,\ldots,n | \varphi_1(A(i)) = r\}} F(q, x_i) =$
$\sum_{\{i \in \{1,\ldots,n | \varphi(\varphi_1(A(i))) = \varphi(r)\}} F(\varphi(q), x_i) = F_2(\varphi(q), \varphi(r))$.
$q = \varphi_1(A(i)), r = \varphi_1(A(j))$ for at least one $i, j$. This is flow between two (former) anchors, so
$F_1(q, r) = F_{\mathcal{L}}(\varphi_1^{-1}(q), \varphi_1^{-1}(r) + (\sum_{\{i,j \in \{1,\ldots,n | \varphi_1^{-1}(A(i)) = q \wedge \varphi_1^{-1}(A(j)) = r\}\}} F(x_i, x_j)) =$
$F_{\mathcal{L}}(\varphi_2^{-1}(\varphi(q)), \varphi_2^{-1}(\varphi(r)) + (\sum_{\{i,j \in \{1,\ldots,n | \varphi_2^{-1}(A(i)) = q \wedge \varphi_2^{-1}(A(j)) = r\}\}} F(x_i, x_j)) =$
$F_2(\varphi(q), \varphi(r))$. Thus, $\psi$ is an isomorphism such that $D_1 \equiv_\psi D_2$. ■

We will now show that when adding multiple links to distinct elements, the order in which we add links does not matter.

**Lemma 6.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN. Let $\sigma, \rho \in (P \cup T)^*$ be such that $elts(\sigma) \cap elts(\rho) = \emptyset$. Let $L_1 = \mathbb{L}(\pi(\sigma))$, let $L_2 = \mathbb{L}(\pi(\rho))$. Let $\varphi_1, \varphi_2$ be two isomorphisms of $L_1$ and $L_2$ respectively, such that $\mathcal{D}, \varphi_1(L_1)$, and $\varphi_2(L_2)$ are mutually disjoint. Then:*

$\lambda(\lambda(\mathcal{D}, (x_1, \ldots, x_n), \varphi_1), (z_1, \ldots, z_k), \varphi_2) = \lambda(\lambda(\mathcal{D}, (z_1, \ldots, z_k), \varphi_2), (x_1, \ldots, x_n), \varphi_1)$

*Proof.* Let $D_1 = \lambda(\lambda(D, \rho, \varphi_2), \sigma, \varphi_1)$, $D_2 = \lambda(\lambda(D, \sigma, \varphi_1), \rho, \varphi_2)$, $D_i = (P_i, T_i, F_i, \pi_i)$, and $L_i = (P_{L_i}, T_{L_i}, F_{L_i}, \pi_{L_i}, A_{L_i})$ for $i \in \{1, 2\}$. We compare $D_1$ and $D_2$.

$P_1 = (P \setminus elts(\rho)) \cup \varphi_2(P_{L_2}) = ((P \setminus elts(\rho)) \cup \varphi_1(P_{L_1}) \setminus \sigma) \cup \varphi_2(P_{L_2}) = ((P \setminus elts(\rho)) \setminus elts(\sigma)) \cup \varphi_1(P_{L_1}) \cup \varphi_2(P_{L_2}) = ((P \setminus elts(\sigma)) \cup \varphi_2(P_{L_2}) \setminus elts(\rho)) \cup \varphi_1(P_{L_1}) = (P \setminus elts(\sigma)) \cup \varphi_1(P_{L_1}) = P_2$.

The reasoning for $T_1 = T_2$ is identical.

The labeling functions $\pi_1$ and $\pi_2$ have the same domain. Moreover, $P_1 \cup T_1 = P_2 \cup T_2 = (P \cup T) \setminus (elts(\rho) \cup elts(\sigma)) \biguplus \varphi_1(P_1 \cup T_1) \biguplus \varphi_2(P_2 \cup T_2)$,
and $\pi_1(z) = \pi_2(z) = \pi(z)$ if $z \in (P \cup T) \setminus (elts(\rho) \cup elts(\sigma))$,
$\pi_1(z) = \pi_2(z) = \pi_L(\varphi_1^{-1}(z)$ if $z \in \varphi_1(P_{L_1} \cup T_{L_1})$, and
$\pi_2(z) = \pi_1(z) = \pi_L(\varphi_2^{-1}(z)$ if $z \in \varphi_2(P_{L_2} \cup T_{L_2})$.
Hence $\pi_1 = \pi_2$.

Finally we compare $F_1$ and $F_2$. Let $(a, b) \in (P_1 \times T_1) \cup (T_1 \times P_1)$. If $a, b \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$, then $F_1(a, b) = F_2(a, b) = F(a, b)$.
If $a, b \in \varphi_i(L_i)$, then $F_1(a, b) = F_2(a, b) = F_{L_i}(\varphi_i^{-1}(a)), \varphi_i^{-1}(b))$.
If $a \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$ and $b = \varphi_1(A_1(j))$ for some $j \in \{1, \ldots, n\}$, then $F_1(a, b) = F(a, x_j) = F_2(a, b)$.
And similarly for the three remaining cases with
$a \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$ and $b = \varphi_2(A_2(j))$ for some $j \in \{1, \ldots, n\}$,
$a = \varphi_1(A_1(j))$ for some $j \in \{1, \ldots, n\}$ or $a = \varphi_2(A_2(j))$ for some $j \in \{1, \ldots, n\}$ and $b \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$. Thus $D_1 = D_2$. ■

Finally we show that multiple linking operations can be done at the same time.

**Lemma 7.** *Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN. Let $\sigma, \rho \in (P \cup T)^*$ be such that $elts(\sigma) \cap elts(\rho) = \emptyset$. Let $\mathcal{L}_1 = \mathbb{L}(\pi(\sigma))$, $\mathcal{L}_2 = \mathbb{L}(\pi(\rho))$. Let $\varphi_1, \varphi_2$ be isomorphisms of $\mathcal{L}_1$, and $\mathcal{L}_2$ respectively, such that $\mathcal{D}, \varphi_1(\mathcal{L}_1)$, and $\varphi_1(\mathcal{L}_2)$ are mutually disjoint. Let $\mathcal{L}_3 = \mathcal{L}(\pi(\sigma \circ \rho)) \equiv_\psi \varphi_1(\mathcal{L}_1) \cup \varphi_2(\mathcal{L}_2)$. Let $\varphi_3$ be an isomorphism for $\mathcal{L}_3$ such that for all elements $z$ of $\mathcal{L}_3$, $\varphi_3(z) = \varphi_1(z)$ if $\psi^{-1}(z)$ an element of $\mathcal{L}_1$, and $\varphi_3(z) = \varphi_2(z)$ if $\psi^{-1}(z)$ an element of $\mathcal{L}_2$. Then $\lambda(\lambda(\mathcal{D}, \sigma, \varphi_1), \rho, \varphi_2) = \lambda(\mathcal{D}, \sigma \circ \rho, \varphi_3)$*

*Proof.* Let $D_1 = \lambda(\lambda(D, \rho, \varphi_2), \sigma, \varphi_1)$, $D_2 = \lambda(\lambda(D, \sigma, \varphi_1), \rho, \varphi_2)$, $D_i = (P_i, T_i, F_i, \pi_i)$ for $i \in \{1, 2\}$. We use a similar proof as for Lemma 6.

$P_1 = ((P \setminus elts(\sigma)) \setminus elts(\rho)) \cup \varphi_1(P_{\mathcal{L}_1}) \cup \varphi_2(P_{\mathcal{L}_2}) = (P \setminus elts(\sigma \circ \rho)) \cup \varphi_3(P_{\mathcal{L}_1}) \cup \varphi_3(P_{\mathcal{L}_2}) = (P \setminus elts(\sigma \circ \rho)) \cup \varphi_3(P_{\mathcal{L}_1} \cup P_{\mathcal{L}_2}) = (P \setminus elts(\sigma \circ \rho)) \cup \varphi_3(P_{\mathcal{L}_3} = P_2$

The reasoning for $T_1 = T_2$ is identical.

The labeling functions $\pi_1$ and $\pi_2$ have the same domain. Moreover, $P_1 \cup T_1 = P_2 \cup T_2 = (P \cup T) \setminus (elts(\rho) \cup elts(\sigma)) \biguplus \varphi_1(P_1 \cup T_1) \biguplus \varphi_2(P_2 \cup T_2)$.
$\pi_1(z) = \pi_2(z) = \pi(z)$ if $z \in (P \cup T) \setminus (elts(\rho) \cup elts(\sigma))$,
$\pi_1(z) = \pi_{\mathcal{L}_1}(\varphi_1^{-1}(z) = \pi_{L_3}(\psi(\varphi_1^{-1}(z))) = \pi_2(z)$ if $z \in \varphi_1(P_{L_1} \cup T_{\mathcal{L}_1})$, and
$\pi_1(z) = \pi_{\mathcal{L}_2}(\varphi_2^{-1}(z) = \pi_{L_3}(\psi(\varphi_2^{-1}(z))) = \pi_2(z)$ if $z \in \varphi_1(P_{L_2} \cup T_{\mathcal{L}_2})$. Hence $\pi_1 = \pi_2$.

Finally we compare $F_1$ and $F_2$. Let $(a, b) \in (P_1 \times T_1) \cup (T_1 \times P_1)$. If $a, b \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$, then $F_1(a, b) = F_2(a, b) = F(a, b)$.
If $a, b \in \varphi_i(\mathcal{L}_i)$, then $F_1(a, b) = F_{\mathcal{L}_i}(\varphi_i^{-1}(a)), \varphi_i^{-1}(b)) = F_3(\psi(\varphi_i^{-1}(a), \psi(\varphi_i^{-1}(b)) = F_2(a, b)$
If $a \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$ and $b = \varphi_1(A_1(j))$ for some $j \in \{1, \ldots, n\}$, then $F_1(a, b) = F_{\mathcal{L}_1}(a, x_j) = F_{\mathcal{L}_3}(\psi(a), \psi(b))$.

And similarly for the three remaining cases with
$a \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$ and $b = \varphi_2(A_2(j))$ for some $j \in \{1, \ldots, n\}$,
$a = \varphi_1(A_1(j))$ for some $j \in \{1, \ldots, n\}$ or $a = \varphi_2(A_2(j))$ for some $j \in \{1, \ldots, n\}$
and $b \in ((P \cup T) \setminus elts(\sigma)) \setminus elts(\rho)$. Thus $D_1 = D_2$. ■

*Example 7.* In this example we will add two links to the net $\mathcal{D}$ in Figure 10, and show how, per Lemma 7, we can also do it with one link.

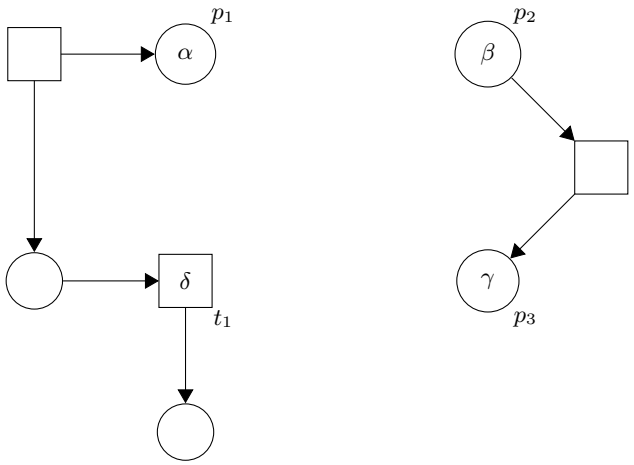


**Fig. 10.** A net $\mathcal{D}$ consisting of two components that will be linked with channels.

We would like to end with a net that looks like the net in Figure 11:

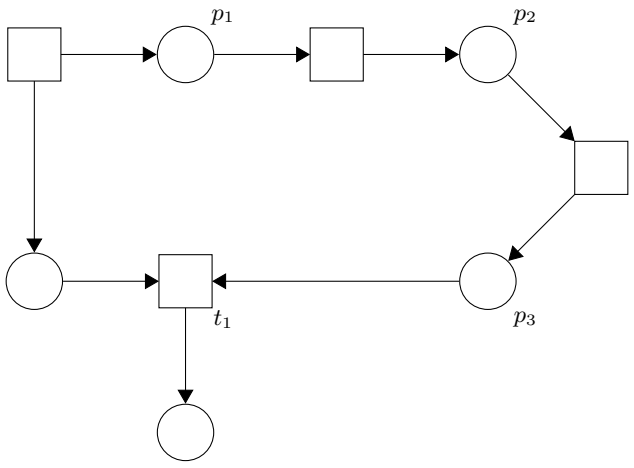


**Fig. 11.** The net structure we want.

We have the following three protocol nets:

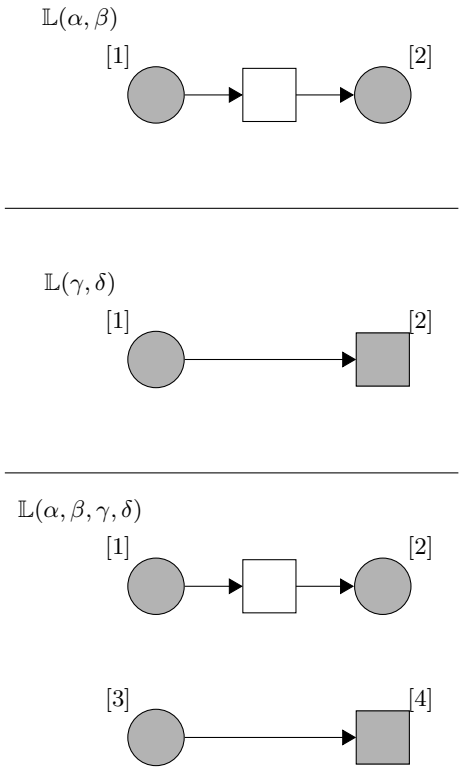


**Fig. 12.** Three protocol nets

We note that $\mathbb{L}(\alpha, \beta, \gamma, \delta) = \mathbb{L}(\alpha, \beta) \cup \mathbb{L}(\gamma, \delta)$.

Now we can either link $\lambda(\lambda(\mathcal{D}, (p_1, p_2)), (p_3, t_1))$ or we can link $\lambda(\mathcal{D}, (p_1, p_2, p_3, t_1))$ for the same result.

## 5 Generating Petri nets

To demonstrate the versatility of protocol nets, we will now show how to create any given Petri net from a DIN with a single place and a single transition using protocol nets.

Let $\alpha, \beta$ be protocol labels such that $\alpha \in \Pi_P$ and $\beta \in \Pi_T$. Let $\mathbb{L}_u$ be a protocol assignment function with $\mathbb{L}_u(\alpha), \mathbb{L}_u(\beta), \mathbb{L}_u(\alpha, \beta), \mathbb{L}_u(\beta, \alpha)$ as given in Figure 13.

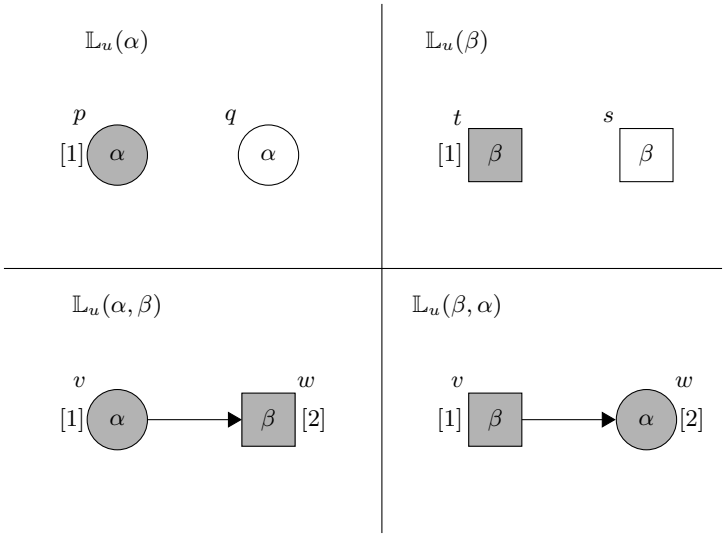


**Fig. 13.** The protocol function $\mathbb{L}_u$.

The protocol nets $\mathbb{L}_u(\alpha)$ and $\mathbb{L}_u(\beta)$ will be used to generate places and transitions, respectively. $\mathbb{L}_u(\alpha,\beta)$, and $\mathbb{L}_u(\beta,\alpha)$ will be used to generate flow. Let $D$ be the following DIN:

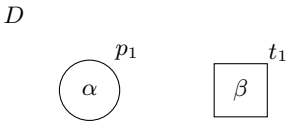


**Fig. 14.** The initial DIN $D$.

**Theorem 1.** *Let $N = (P, T, F)$ where $P \neq \emptyset$ and $T \neq \emptyset$ be a Petri net. There exists a DIN $\hat{D}$ with underlying net isomorphic to $N$, such that $\hat{D}$ is the result of adding $|P| + |T| + (\sum_{(a,b)\in DOM(F)} F(a,b)) - 2$ links to $D$.*

*Proof.* Let $N = (P, T, F)$ be the net structure we wish to create. We first add places to $D$ using the $\mathbb{L}_u(\alpha)$ protocol net. For all $i \in \{2, 3, ..., |P|\}$ we define $\psi_i$ as an isomorphism of $\mathbb{L}_u(\alpha)$ such that $\psi(p) = p_i$ and the $\psi_i(\mathbb{L}_u(\alpha))$ are mutually disjoint and disjoint with $D$. For all $i \in \{3, ..., |P|\}$ we define $D_i = \lambda(D_{i-1}, (p_{i-1})), \psi_i)$, with $D_2 = \lambda(D, (p_1), \psi_2)$. Each DIN $D_i$ has one more place than $D_{i-1}$. Because in each $D_i$, place $p_i$ is labeled by $\alpha$, it is guaranteed that in each step the protocol net $\mathbb{L}_u(\alpha)$ will be used. Thus $D_{|P|}$ is a DIN with $|P|$ places and 1 transition.

Now we add the transitions. For all $j \in \{2, 3, ..., |T|\}$ we define $\tau_j$ as an isomorphism of $\mathbb{L}_u(\beta)$ such that $\tau_j(t) = t_j$ and the $\tau_j(\mathbb{L}_u(\beta))$ are mutually disjoint and disjoint with $D$. For all $j \geq 3$ we define $D'_j = \lambda(D'_{j-1}, (t_{j-1}), \tau_j)$, with $D'_2 = \lambda(D_{|P|}, (t_1), \tau_2)$. Just like with the places, each DIN $D'_j$ has one more transition than $D'_{j-1}$. Thus $D'_{|T|}$ is a DIN with $|P|$ places and $|T|$ transitions.
To add the arcs with their weights, we assume given a bijection $\varphi_0$ from $P \cup T$, the elements of $N$, to the elements of $D'_{|T|}$, such that places map to places and transitions map to transitions. As both nets have the same number of places and

transitions, such a bijection exists.
Let $NDOM(F) = \{(x,y) \in \mathrm{DOM}(F) | F(x,y) > 0\}$ be the *non-trivial domain of* $F$. Let $\#F = |NDOM(F)|$. Let $(x_1,y_1),\ldots,(x_{\#F},y_{\#F})$ be an enumeration of all elements of $NDOM(F)$. For all $(x_i,y_i) \in NDOM(F)$, we introduce distinct new places/transitions as follows:
$x_i^j$ of the same type as $x_i$, $1 \leq j \leq F(x_i,y_i)$,
$y_i^j$ of the same type as $y_i$, $1 \leq j \leq F(x_i,y_i)$,
For all $i \in \{1,\ldots,\#F\}$ and all $j \in \{1,\ldots,F(x_i,y_i)\}$, let $\psi_{i,j}$ be an isomorphism for $\mathbb{L}(\pi(\varphi_0(x_i),\pi(\varphi_0(y_i))$ such that for the elements $v,w$ in $(\mathbb{L}_u(\alpha,\beta)$ and $(\mathbb{L}_u(\beta,\alpha)$, $\psi_{i,j}(v) = x_i^j$ and $\psi_{i,j}(w) = y_i^j$.
For each $i \in \{1,\ldots,\#F\}$, each $\hat{D}_{i,j}$ has flow $j$ between $\varphi_i(x_i)$ and $\varphi_i(y_i)$. In particular, for all $i$ and for all $k \in \{1,\ldots,i\}$ $\hat{D}_{i,F(x_i,y_i)}$ has the flow $F(x_k,y_k)$ between $\varphi(x_k)$ and $\varphi(y_k)$. We use $\psi_{i,j}$ and $\varphi_i$ to keep track of the $x_i$ and $y_i$, as they are renamed in each link. When we reach $\hat{D}_{i,F(x_i,y_i)}$, we update the $\varphi_i$ bijection to reflect the new names of $x_i$ and $y_i$, so the next iteration can address the correct elements:
Let $\hat{D}_{1,0} = D'_{|T|}$. For all $i \in \{1,\ldots,\#F\}$ and all $j \in \{2,\ldots,F(x_i,y_i)\}$, $\hat{D}_{i,j} = \lambda(\hat{D}_{i,j-1},(x_i^{j-1},y_i^{j-1}),\psi_{i,j})$.
For all $i \in \{1,\ldots,\#F\}$, we define $\varphi_i : (P \cup T) \to (P_{\hat{D}_{i,F(x_i,y_i)}} \cup T_{\hat{D}_{i,F(x_i,y_i)}})$ as follows:
$\varphi_i(x_i) = x_i^{F(x_i,y_i)}, \varphi_i(y_i) = y_i^{F(x_i,y_i)}$, and $\varphi_i(z) = \varphi_{i-1}(z)$ for all $z \in (P \cup T) \setminus \{x_i,y_i\}$.
For all $i \in \{1,\ldots,\#F\}$, $\hat{D}_{i,1} = \lambda(\hat{D}_{i,0},(\varphi_{i-1}(x_i),\varphi_{i-1}(y_i)),\psi_{i,1})$. For $i \geq 2, \hat{D}_{i,0} = \varphi_i(\hat{D}_{i-1,F(x_i,y_i)})$.
Let $\hat{D} = \hat{D}_{\#F,F(x_{\#F},y_{\#F})}$. Then $\varphi_{\#F}$ is an isomorphism from the elements of $N$ to the underlying net of $\hat{D}$. We know it is bijective because we did not add more elements since $\hat{D}_{1,0}$, and we have added every arc and their weight correctly. ■

*Example 8.* Consider the Petri net $N$ depicted in Figure 15.

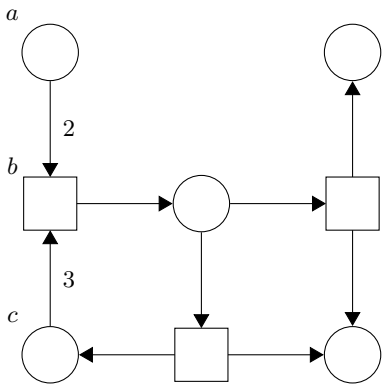


**Fig. 15.** The Petri net $N$.

To construct $N$ as described in the proof of Theorem 1, we first create successively $\lambda(D_{i-1},(p_{i-1}),\psi_i)$ for $i = 2,\ldots,5$, resulting in the net $D_5$ with 5 places and one transition.

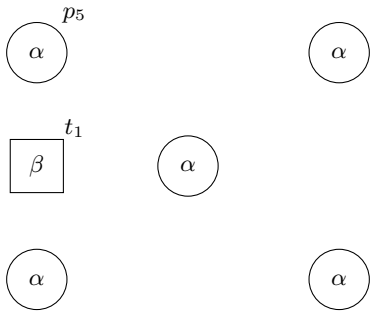


**Fig. 16.** $D_{|P|}$ has 5 places, just like $N$.

Next we obtain $\lambda(D'_{j-1}, (t), \tau_j)$ for $j = 2, 3$ to add the transitions. Once we have all elements, we assume a bijection $\varphi_0$ to match each element in $N$ to one in $D_{|T|}$:

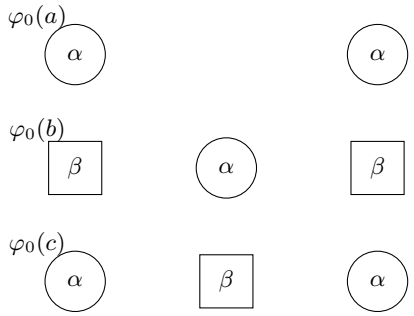


**Fig. 17.** $D_{|T|}$ has 5 places and 3 transitions, just like $N$.

Finally we add the arcs. Let's start with the arc with weight 2 between $a$ and $b$, so $x_1 = a, y_1 = b$. Then we create $\hat{D}_{1,1} = \lambda(\hat{D}_{i,0}, (\varphi_0(a), \varphi_0(b)), \psi_{1,1})$. The elements $\varphi_0(a)$ and $\varphi_0(b)$ are overwritten by the elements $x, y \in \mathbb{L}_u(\alpha, \beta)$, which are renamed by $\psi_1^1$ to $a_1^1$ and $b_1^1$.

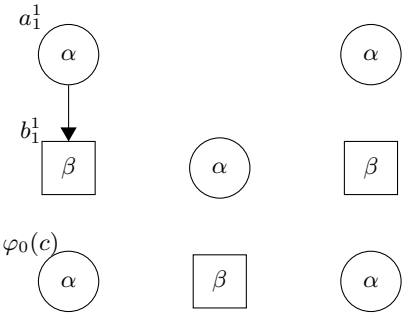


**Fig. 18.** $\hat{D}_{1,1}$ has an arc with weight 1 between the nodes corresponding to $a$ and $b$.

Now we apply the same protocol net to $a_1^1$ and $b_1^1$, adding their weights so the resulting weight is 2:

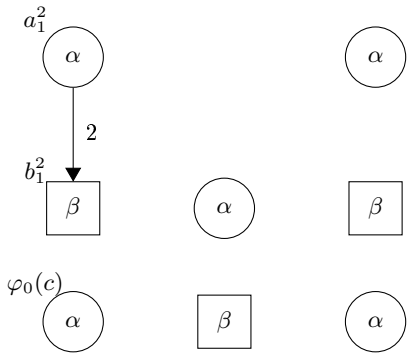


**Fig. 19.** $\hat{D}_{1,2}$ has an arc with weight 2 between the nodes corresponding to $a$ and $b$.

Now we move on to the next arc. To keep correctly addressing elements, we use the mapping $\varphi_1$ which maps $a$ to $a_1^2$ and $b$ to $b_1^2$.

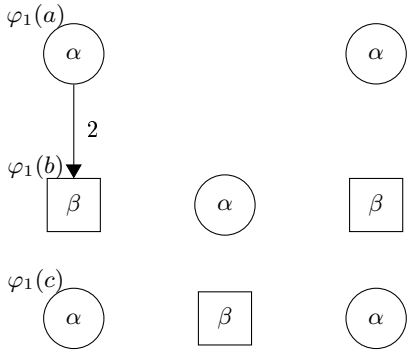


**Fig. 20.** $\hat{D}_{2,0}$ has a completed first arc.

Next we add the arc with weight 3 between $c$ and $b$:

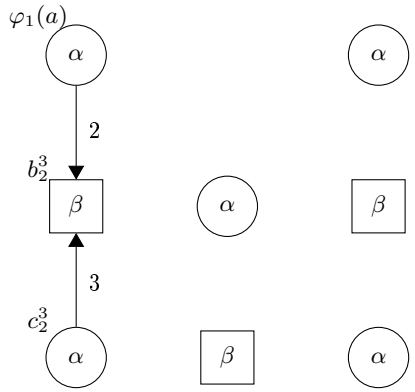


**Fig. 21.** $\hat{D}_{2,3}$ has an arc with weight 3 between the equivalent of $c$ and $b$.

And continuing in this way, we add all arcs:

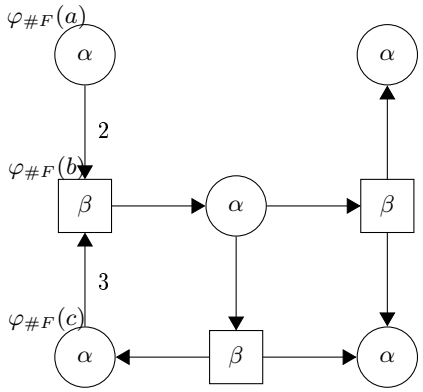


**Fig. 22.** $\hat{D}$ is isomorphic to $N$.

■

## 6 Composition by Merging and Channels

Many papers have been written on composition of Petri nets, using various different methods of composing. These papers often use variants of Petri nets with additional information or behavior to the elements. However, on a net-structural level we can simulate these different methods using protocol nets to create DINs that are isomorphic to their results.

There is more to these papers than just their choice of net structure for composition. Their main focus tends to be on deciding which elements are connected in the first place. This is most often done using one of two methods: First, by giving elements of different nets the same name, so they are merged when taking the union of the two nets. Secondly, putting labels on elements, such as input and output labels on different ends of a channel. Our protocol linking framework does not prescribe a specific method for selection of elements to be linked, so any method can easily be adopted.

We can create DINs with underlying nets that are isomorphic to the component nets used in these papers. Then, using specific protocol nets, we can create linked nets that are isomorphic to the composed system in the original paper.

### 6.1 Composition by Identification

In [6], Petri nets are composed by connecting 'input' and 'output' places in order to study the behavior of the composed system. This is done with a composition method based on so-called *components*. A component is a Petri net with two separate disjoint subsets of places: $I$ and $O$, the component's *input* and *output places* respectively. Places which are neither input nor output are referred to as *internal places*. Two components are composed by taking the union of their elements and arcs. The input places of one component may be the same as the output places of the other, and these places are merged into one place.

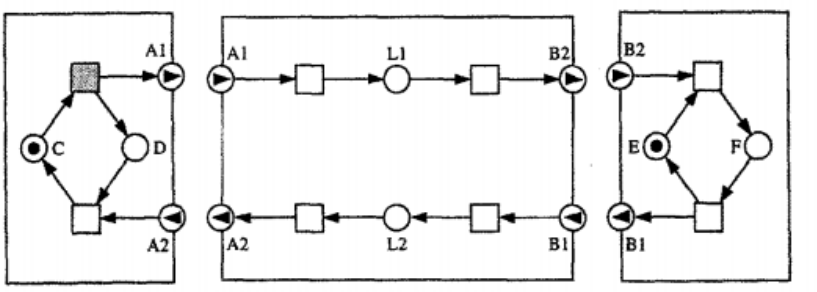


**Fig. 23.** Image taken from [6], showing three components. Places with an arrow pointing into the component are input places, places with an arrow pointing outside of the component are output places.

We now briefly summarize the definitions from [6] for components and composition:

**Definition 16.** *1) A* component *is a tuple* $\Gamma = (P, T, F, I, O)$ *where* $(P, T, F)$ *is a Petri net,* $I, O \subseteq P$, $I \cap O = \emptyset$, ${}^\bullet T \cap O = \emptyset$, *and* $T^\bullet \cap I = \emptyset$.
*2) For* $i \in \{1, 2\}$, *let* $\Gamma_i = (P_i, T_i, F_i, I_i, O_i)$ *be a component. Then* $\Gamma_1$ *and* $\Gamma_2$ *are* composable *if* $P_1 \cap P_2 = (I_1 \cap O_2) \cup (I_2 \cap O_1)$ *and* $T_1 \cap T_2 = \emptyset$.
*3) For two composable components* $\Gamma_1$ *and* $\Gamma_2$, *the* composed system *is* $\Gamma_1 \square \Gamma_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), (O_1 \setminus I_2) \cup (O_2 \setminus I_1))$. ■

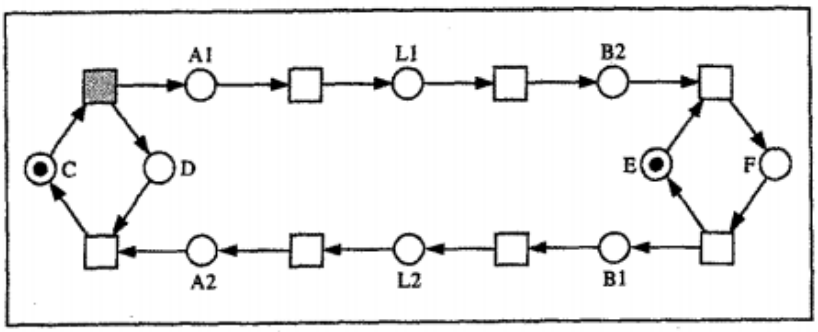


**Fig. 24.** The three components composed into one net with no more input or output places remaining.

Many other papers use place identification/merging as their method of composition. [10] uses shared buffer places between its components. Place merging is used in [17] and [15]. One notable example is [13] for its more complicated method, using indices and labels, for deciding which places are merged in what order. In this paper we simulate the composition method from [6] as illustration for how protocol nets can be used in general for simulating composition by identification.

The components used in [6] have places with the same name, in order for them to be merged in the composition. On the other hand, linking Petri nets (DINs) using protocol nets assumes that the nets involved are disjoint. However, we can easily rename the shared input and output places and then link compatible pairs of input places and output places. We first present the protocol nets we will be using. It should be noted that these protocol nets can be used for any method of composition by identification, not just Kindler's method:

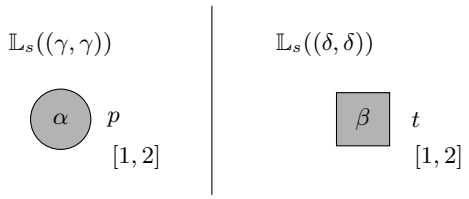


**Fig. 25.** Two protocol nets: The left for merging two places, the right for merging two transitions.

$\mathbb{L}_s((\gamma,\gamma))$ and $\mathbb{L}_s((\delta,\delta))$ are 2-sided protocol nets where both anchors are the same element. We now show how to use these protocols to simulate the composition method of [6]:

**Theorem 2.** *For $i \in \{1,2\}$ let $\Gamma_i = (P_i, T_i, F_i, I_i, O_i)$ be two composable components. Let $\mathcal{D}_1, \mathcal{D}_2$ be two DINs where the underlying net of $\mathcal{D}_1$ is isomorphic with $\Gamma_1$ and the underlying net of $\mathcal{D}_2$ is isomorphic with $\Gamma_2$. Then a series of protocol links on $\mathcal{D}_1 \cup \mathcal{D}_2$ will create a DIN whose underlying net is isomorphic with $\Gamma_1 \square \Gamma_2$.*

*Proof.* For $i \in \{1,2\}$ let $\mathcal{D}_i = (P'_i, T'_i, F'_i, \pi_i)$ be two disjoint DINs isomorphic via $\varphi_i$ to $\Gamma_i$. Let $\mathcal{D}'_0 = (P, T, F, \pi) = \mathcal{D}_1 \cup \mathcal{D}_2$. For all places $p \in P$, $\pi(p) = \gamma$. $\mathbb{L}_s((\gamma,\gamma))$ is place-identification protocol net from Figure 25. Let $x_1, \ldots, x_k$ be an ordering for all elements $x_j \in ((I_1 \cap O_2) \cup (I_2 \cap O_1))$. For each $j \in \{1, \ldots, k\}$, let $i_j, o_j \in P$ be $\varphi_1(x_j)$ and $\varphi_2(x_j)$ respectively. We create a new DIN $\mathcal{D}'_j := \lambda(\mathcal{D}'_{j-1}, (i_j, o_j), \varphi_{x_j})$. Because each $D'_j$ merges one pair of places in the same way $\square$ does, $\mathcal{D}'_k$ is isomorphic with $\Gamma_1 \square \Gamma_2$ as it has paired up all applicable inputs and outputs. ■

Note that per Lemma 6, the order in which we pick the $(i_j, o_j)$ does not matter, and per Lemma 7 we could do all links in one go with a big enough protocol net.

### 6.2 Asynchronous Composition via Channels

Another very common composition method is to use *channels*. Instead of merging elements, a channel adds a new element 'between' them, with one-directional flow. This allows for a more indirect connection between subsystems.

An example of a paper using channels is [7], which composes 'enterprise nets' into an 'industry net' using different labels representing various types of messages between enterprises in an industry.

In [4], the composition of *modal Petri nets* is discussed. In modal Petri nets transitions are either *may* or *must*, which determines which of the transitions may or must fire at a given marking. As this thesis is focused on net structures, not firing sequences, this may-must distinction is not relevant for our thesis. We are simply interested in the method they use to compose these nets: transition-to-transition channels with a buffer place. It is a good example of a method of composition that our protocol nets can easily emulate on the net-structural level while additional information such as modality can be added without disrupting the protocol link framework. As in the previous section, we will use the method in [4] as an illustration for how to simulate channel composition in general.

We introduce a simplified version of the nets and composition in [4], cutting out the modality and firing rule parts:

**Definition 17.** *1) An* I/O Alphabet $\Sigma$ *is an alphabet partitioned into disjoint sets* in, out *and* int *of input, output, and internal labels respectively. Two I/O alphabets* $\Sigma_1, \Sigma_2$ *are* composable *if* $in_1 \cap in_2 = out_1 \cap out_2 = \Sigma_1 \cap int_2 = \Sigma_2 \cap int_1 = \emptyset$.

*2) A* labeled Petri net over an I/O alphabet $\Sigma$ *is a tuple* $N = (P, T, F, \lambda)$ *where* $(P, T, F)$ *is a Petri net, and* $\lambda : T \to \Sigma \cup \{()\}$ *is a* transition labeling function *where* $()$ *denotes the empty sequence.*

*3) Let* $N_1 = (P_1, T_1, F_1, \lambda_1)$ *and* $N_2 = (P_2, T_2, F_2, \lambda_2)$ *be labeled Petri nets over* $\Sigma_1, \Sigma_2$ *respectively. Let* $\Sigma_1$ *and* $\Sigma_2$ *be composable, and* $(P_1, T_1, F_1)$ *and* $(P_2, T_2, F_2)$ *be disjoint. Let for each* $a \in \Sigma_1 \cap \Sigma_2, p_a$ *be a new place. Then the* asynchronous composition $N = N_1 \bigotimes N_2 = (P, T, F, \lambda)$ *is the labeled Petri net defined as follows:* $P = P_1 \cup P_2 \cup \{p_a | a \in \Sigma_1 \cap \Sigma_2\}$, $T = T_1 \cap T_2$, *for all* $x, y \in P_1 \cup P_2 \cup T_2 \cup T_2$, $F(x,y) = F_1(x,y)$ *if* $x, y \in P_1 \cup T_1$ *and* $F_2(x,y)$ *otherwise; for all* $\{i,j\} = \{1,2\}$ *and for each* $t \in T_i$ *and* $p_a \in P$ *with* $a \in \Sigma_1 \cap \Sigma_2$, $F(t, p_a) = 1$ *if* $a = \lambda_i(t) \in in_j \cap out_i$, *and 0 otherwise.* $F(p_a, t) = 1$ *if* $a = \lambda_i(t) \in in_i \cap out_j$, *and 0 otherwise.*

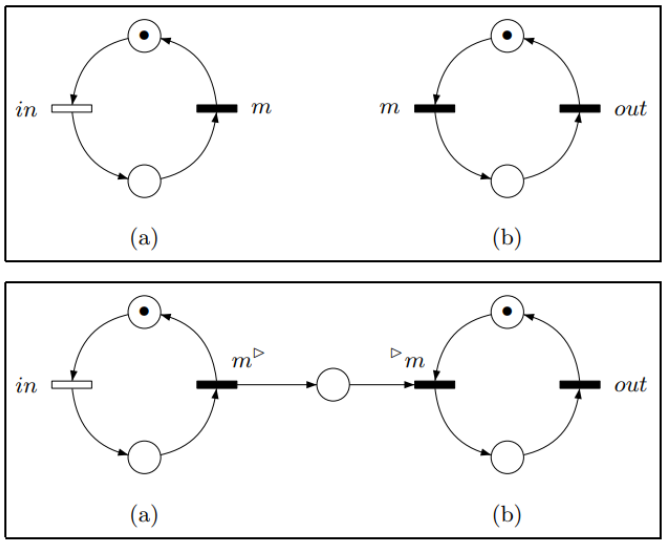


**Fig. 26.** Two labeled Petri nets and their composition.

Just like in the previous section, we can also emulate asynchronous compositions with protocol nets. We first define *channel protocol nets* in Figure 27. To emulate modal nets we only use the transition-to-transition channel, but the other channel types are also useful for other methods.

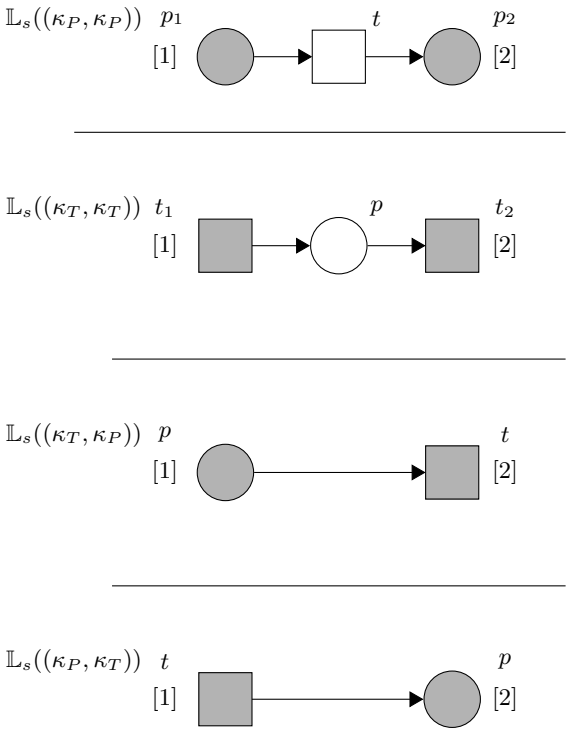


**Fig. 27.** Four different protocol nets for connecting two transitions, two places, or transitions and places.

We use two labels for these protocols: $\kappa_P$ and $\kappa_T$, representing a place and a transition wanting to form a channel respectively. Note that the ordering of the anchors matters as well: tokens flow from the first anchor to the second anchor.

**Theorem 3.** *For $i \in \{1,2\}$ let $N_i = (P_i, T_i, F_i, \lambda_i)$ be two labeled Petri nets over composable I/O alphabets $\Sigma_i$ such that $(P_1, T_1, F_1)$ is disjoint with $(P_2, T_2, F_2)$. There exists a DIN $\mathcal{D} = (P, T, F, \pi)$ whose underlying net is isomorphic with $N_1 \cup N_2$ such that a series of protocol links on $\mathcal{D}$ will create a DIN whose underlying net is isomorphic with $N_1 \bigotimes N_2$.*

*Proof.* Let $\mathcal{D} = (P, T, F, \pi)$ be a DIN whose underlying net is isomorphic via $\varphi$ to $N_1 \cup N_2$. For all transitions $t \in T$, $\pi(t) = \kappa_T$. Let $x_1, \ldots, x_k$ be an ordering for all elements $x_i \in \Sigma_1 \cap \Sigma_2$. For each $i \in \{1, \ldots, k\}$, let $u_i, t_i \in T$ be such that $\lambda_1(\varphi^{-1}(u_i)) = \lambda_2(\varphi^{-1}(t_i)) = x_i$. We create a new DIN $\mathcal{D}'_i := \lambda(\mathcal{D}'_{i-1}, (u_i, t_i), \varphi_{x_i})$.

$\mathcal{D}'_i$ has $i$ channels added just like the asynchronous composition method. Thus the last DIN $\mathcal{D}'_k$ is isomorphic with $N_1 \bigotimes N_2$. ■

## 7 Case Studies

In the previous section we presented ways of simulating other methods of composition. But protocol nets can be used in more creative and practical ways. In this section we will explore some of these uses.

### 7.1 Complex Protocols

Many interaction between different components and subsystems are more complex than can be solved with identification or a channel. Examples are semaphores for mutual exclusion problems, ordering based on priority of tasks and components, or creating queues for many different nets to take advantage of one net's services. This section showcases some more complex protocols that are common in practical net interaction.

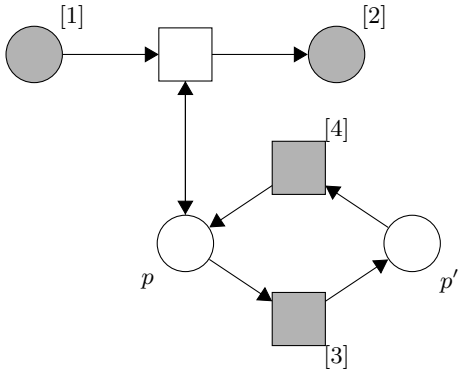


**Fig. 28.** An effector protocol.

Our first complex protocol is in Figure 28: The Effector. Tokens can only flow from $A(1)$ to $A(2)$ if there is a token in $p$. By firing the transitions $A(3)$ and $A(4)$ the data flow is blocked or allowed respectively. As the transitions are anchors, they can

be combined with transitions in components to make more complex conditions for when data flow is stopped or restarted.

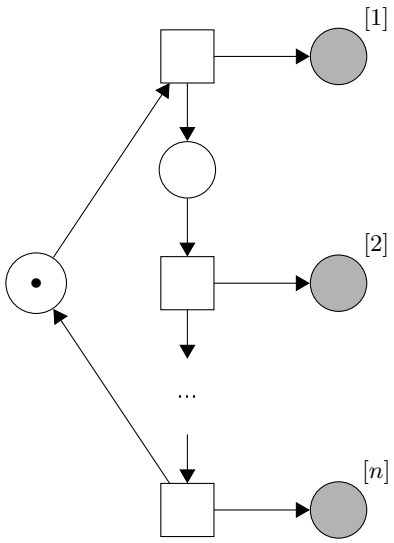


**Fig. 29.** A sequencer protocol.

Our next example is a Sequencer in Figure 29: outputs $A(1), A(2), \ldots, A(n)$ each receive a token in sequence, looping back to $A(1)$ when the last anchor has received a token. It is useful for ensuring different components get an equal amount of tokens, or ensuring that certain components are activated before others. Obviously sequencers can also be adapted to take input in sequence, or a mix of input and output.

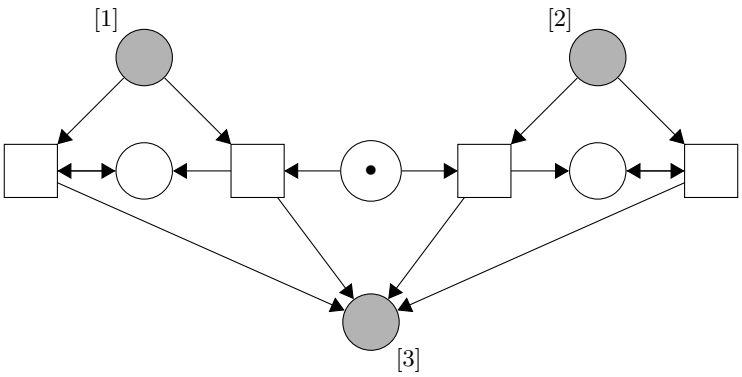


**Fig. 30.** An input selector.

Finally we present the Input Selector: It has two input places $A(1)$ and $A(2)$, and an output place $A(3)$. Once one of $A(1)$ or $A(2)$ has had a token flow to the output $A(3)$, that input is locked in as the only input allowed to keep pushing tokens through the protocol net. This can be used to model a first-come-first-serve situation, for example

a business signing a contract with the first responder, who then gets exclusive rights to keep interacting.

### 7.2 Webservice Nets

When designing a system composed of different components, it is common to express the system's behavior in terms of formal logic or algebra: "Either $A$ or $B$ should be executed." "If $C$ is executed, $D$ must also be executed." "$E$ must always be executed before $F$."

In [5] *web service nets* are proposed. These nets are used to represent online services such as querying databases, purchasing items, digital payment, etc. An example would be a process of wanting to buy an item online. Different stores must be queried whether they sell the item. If there are multiple sellers, they may be compared in some way, and a choice made for which one we buy the item from. Once the store is selected, payment must be made via an online banking service and delivery must be scheduled via a delivery service. Each component in this process would be its own net, and they are linked via various logical operations such as AND, OR, and 'leads to' that are more complicated than the simple channels and identification we have seen before.

We first give a simplified definition of the nets in [5].

**Definition 18.** *A* web service net *is a tuple* $S = (P, T, F, i, o)$ *where* $(P, T, F)$ *is a Petri net and* $i \in P$ *its unique* input place *and* $o \in P$ *its unique* output place *respectively. When* $i$ *is marked with a single token and all other places are empty, the net is said to* start executing. *It is* done executing *when there is a single token in* $o$ *and all other places are empty.*

Web service nets are composed through several *service algebra operators*. Each operator takes nets and expands them with extra elements and flow. For example, Figure 31 shows the OR and AND operations when applied to (abstracted) web service nets. The results are again web service nets with new input and output places.

There are a number of other operations defined in [5], such as sequential operation (S1 must be executed fully before S2 can start), and iteration (S1 executes one or more times). In this case study we will only look at AND, OR, sequence, and iteration. It is easy to verify that the operations work correctly, through basic analysis of the firing sequences of the nets. We can easily make protocol nets that emulate the operations, as seen in Figure 32. The protocol assignment function is of the following form: $\mathbb{L}(\alpha, \beta, \perp, \perp)$, where $\alpha$ indicates what operation is being applied (AND, OR, IT, SEQ), and $\beta$ becomes the label of the input place of the composed net. A special case is the iteration operator, which only acts on one net, so there we use $o$ as the second element.

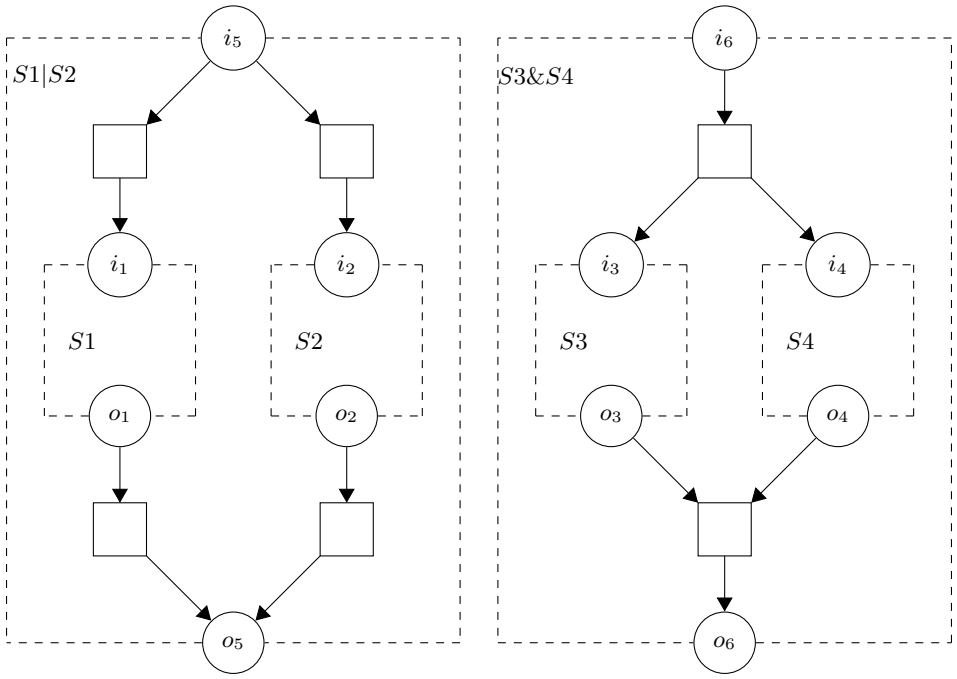


**Fig. 31.** Web service algebra operations for OR and AND.

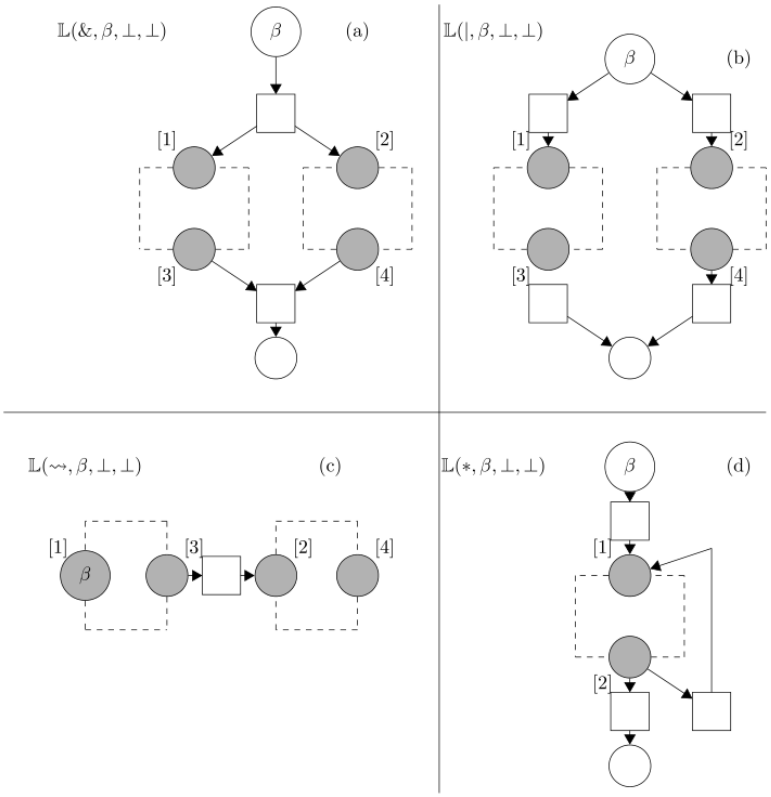


**Fig. 32.** Protocol nets for the operations AND, & (a), OR, | (b), sequence, $\rightsquigarrow$ (c), and iteration, $*$ (d)

Given a set of web service nets and the intended compositional formula, we can make DINs that work with these protocol nets to produce the same composed system. For each web service net $S_k$, we create a DIN $D_k$ whose underlying net is isomorphic with $S_k$. Then we assign the protocol labels as follows: Only the labels of $i_k$ and $o_k$ are relevant, as these are the only elements involved in linking. For most operations, we do not need the labels on the $o_k$ either, so $\pi_k(o_k) = \bot$.

*Example 9.* Let $S_1, \ldots S_4$ be webservice nets, and let the composed system we want be $(S_1|S_2)\&(S_3 \rightsquigarrow (S_4*))$. Then we define the 4 DINs as shown in Figure 33:

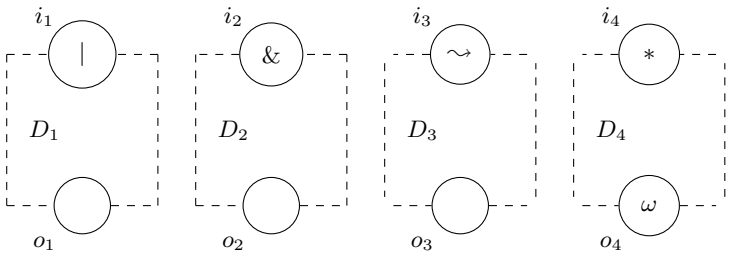


**Fig. 33.** The initial DINs representing the four service nets.

The DIN $D_5 = \lambda(D_1 \cup D_2, (i_1, i_2, o_1, o_2))$ is shown in Figure 34. Note how its input place $i_5$ now has the label &, which was on $i_2$ before. Through this method of storing the label for an operation on one of its children we can create arbitrarily large combinations of the atomic web service nets, because when combining $n$ web service nets, there can only be maximum $n - 1$ operations (apart from iteration, which is pointless to do repeatedly), so each net (except for the very last) stores one operation as a label.

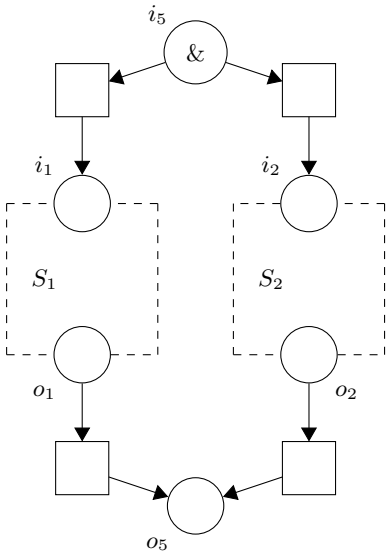


**Fig. 34.** $D_5$ is created by linking the $D_1$ and $D_2$ with the OR protocol.

Creating the link $D_6 = \lambda(D_4, (i_4, o_4))$ creates the DIN in Figure 35. Because $*$ is an operation on a single web service net, $i_6$ gets its label from $o_4$.

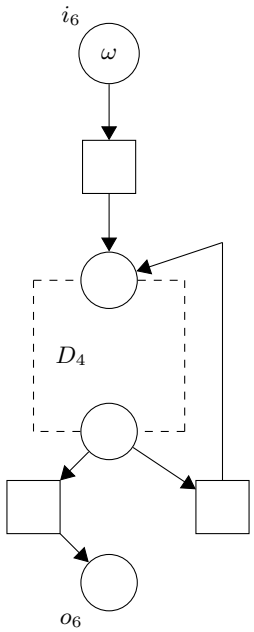


**Fig. 35.** The initial DINs representing the four service nets.

The DIN $D_7 = \lambda(D_4, (i_4, o_4))$ is shown in Figure 36.

Finally, we create the link $D_8 = \lambda(D_5 \cup D_7, (i_5, i_7, o_5, o_7))$ to make the fully composed system seen in Figure 37.

### 7.3 Biology Pathways

Petri net models of biological systems are often difficult to construct and compose. Many systems consist of hundreds of subsystems that have thousands of interactions with each other. On top of that, many interactions are more complicated than can be modeled by the identification or channels we have seen in previous work. A system can have dozens of different types of interactions as well.

A common example of a biological interaction is an effector, as explained in Section 6.3. An effector is a subsystem that must be in a certain state in order for a different subsystem to work. Examples are the presence of a catalyst, the environment having a property such as a certain pressure level or temperature.

The reverse of an effector is an inhibitor. Inhibitors stop another process from functioning if the inhibitor is in a given state. Effector models can easily be turned into inhibitors. To take the example of the effector in Figure 28, assuming that a token in $p$ (and $p'$ empty) signals the presence of a catalyst, simply moving the bidirectional arc on $p$ to $p'$ would make the presence of the catalyst be an inhibitor of the process, rather than an effector.

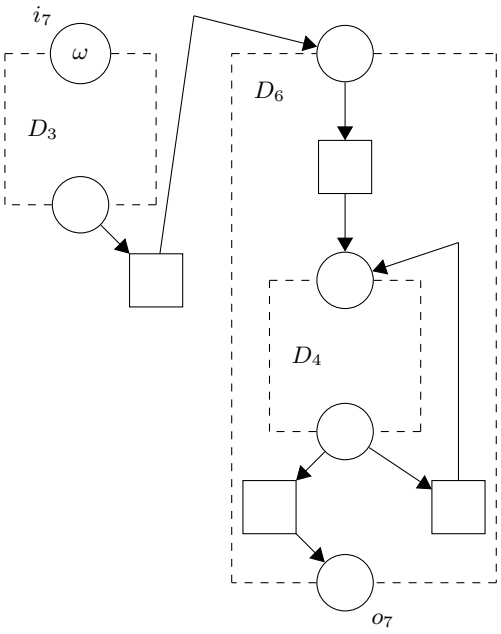


**Fig. 36.** The initial DINs representing the four service nets.

Other common biological interactions are DNA transcription, protein generation, and consumption of energy in the form of ATP, transforming it into ADP.

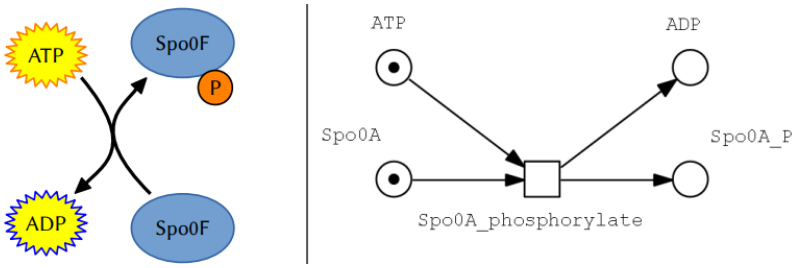


**Fig. 38.** A biological interaction (left) and the Petri net modeling it (right). [3]

There have been attempts by various biologists to list all interactions in a cell [8]. In practice the only viable way to create a realistic Petri net model of such large systems is by creating Petri nets for each subsystem and composing them through their (varied) interactions. For this we can use protocol link nets very flexibly. We already showed the effector, and just with channels and identification can we create an ATP reaction like the one in Figure 38.

We would like to work with biologists to create the best protocol nets and DINs as possible. Biology expertise is crucial for understanding what should be modeled. As we are not biology experts, we would need to work with experts to develop protocol nets based on the latest empirical observations.

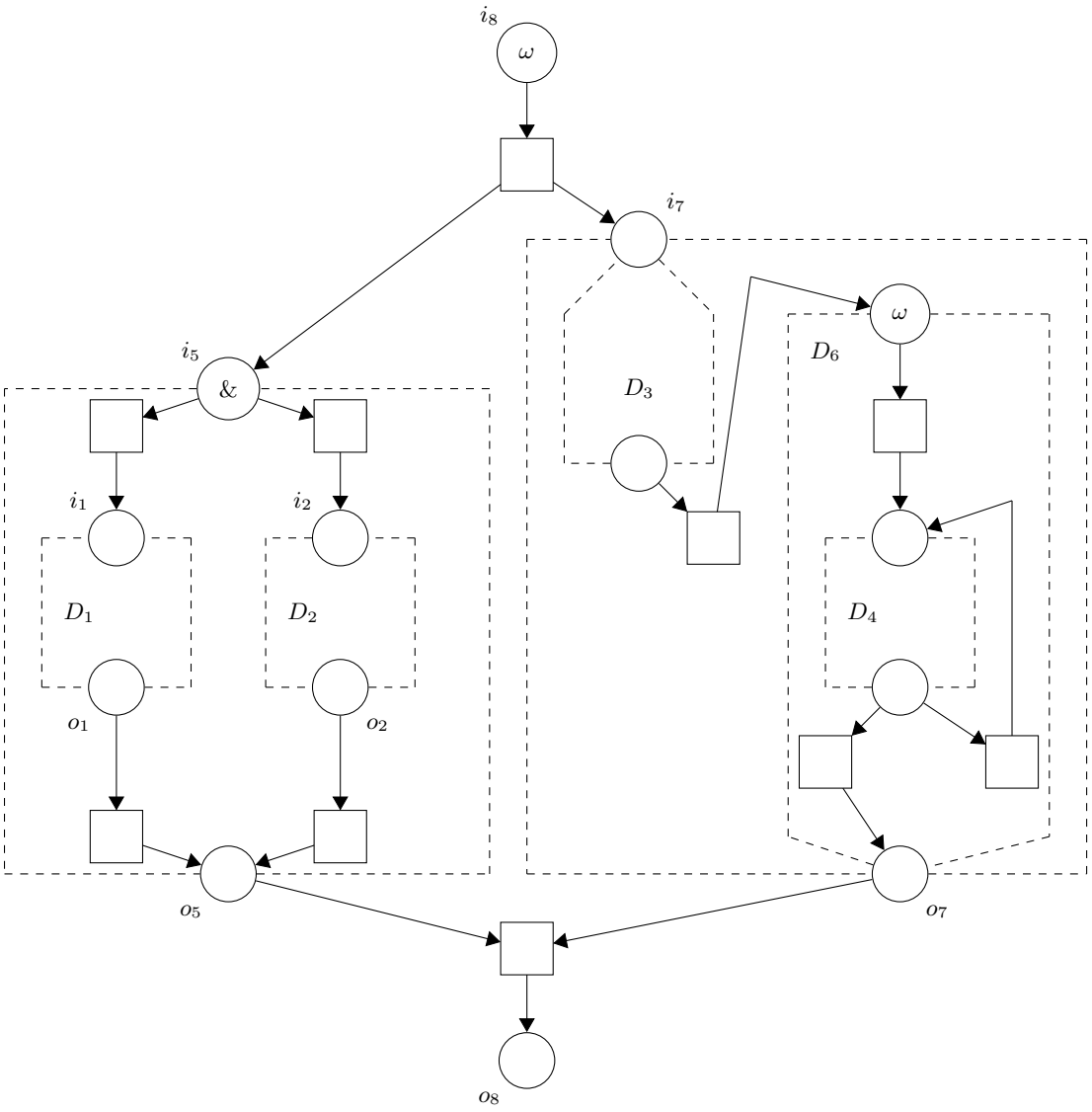


**Fig. 37.** The initial DINs representing the four service nets.

## 8 Related Work

Not all concurrent modeling or composition of systems is done through identification and adding channels, or even with Petri nets. There are other techniques that do similar things to what we present with protocol nets. In this section we mention several of these.

Reo [1] is a coordination language primarily designed for combining concurrent programs and processes into full systems. The design philosophy is very similar to protocol nets: A Reo program (called a circuit) is a labeled directed hypergraph through which data and signals flow. The circuit is places 'between' programs or processes, allowing them to interact indirectly in a complex manner. Where our protocol nets are bound by the well-established rules of Petri nets, in Reo the user can design any formal specifications for edges they want. Because everything is formalized, Reo circuits may be formally verified for correctness. Reo is also designed to be convertible to programming code such as a Java program. It is possible protocol nets may be verified for correctness in a similar manner to Reo, and future work should explore this.

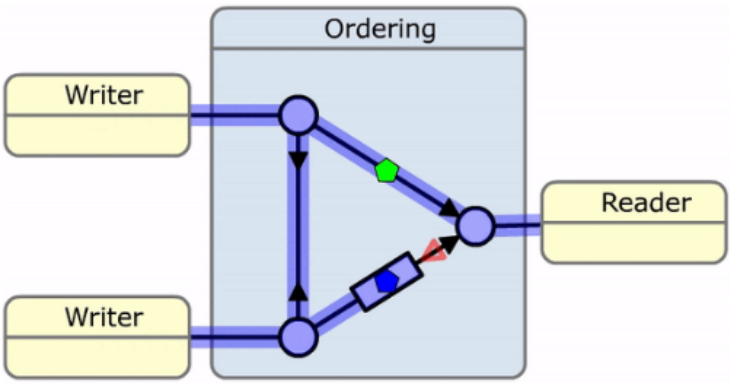


**Fig. 39.** A Reo circuit [1] linking two producers (writers) and a consumer (reader). The consumer receives input from the writers alternatingly.

Graph grammars [14] are a method of generating and combining graphs in a formal, algebraic manner. They work by modifying graphs by applying rewriting rules. Graph grammars are grounded in formal language theory, so there has been much research into them, and there are many variants and types of graph grammars. There are grammars which rewrite single elements into graphs, similar to using a protocol net with one anchor. Other grammars use 'hyperedges', edges connecting more than two nodes, as a basis for rewriting. This is comparable to protocol nets with multiple anchors. Similar to our DINs, graph grammars often use labels on nodes to indicate what kind of operation should be applied. As Petri nets are bipartite graphs, linking with protocol nets can also be seen as a rewriting rule, replacing selected elements with an entire new net. Because of this similarity, it is possible that known results and theorems on graph grammars may also apply to DINs. Although regular graphs lack the firing sequences and other behavioral aspects of Petri nets, there is still a lot of overlap on a structural level. It would be interesting to investigate what graph grammars correspond most to protocols, but

it is beyond the scope of this thesis.

Petri Box calculus [2] is a framework for combining Petri nets in such a way that it doesn't violate algebraic properties. It is focused on Petri 'boxes', subsystems that have their own behavior and properties. It can be compared to the Webservice nets from Section 7.2, with the way nets and their behavior are combined in a formally logical, algebraic fashion. As Petri Box calculus has a lot of results on the behavior of combined nets, it would be a good idea to investigate whether these results can be applied to protocol nets.

# 9 Conclusion and Future Work

In this paper we have introduced a new idea to the discussion on composition of Petri nets: connecting nets by inserting a *protocol net* between them. Unlike many other contributions to composition, our method is not trying to solve one specific problem or tries to describe one type of interaction between components. Instead, it tries to give the user as much freedom as possible to create various types of interactions and links, and makes it easy to combine these different interactions into one framework. We have shown our method can simulate and combine nearly every other method of composition. It should be noted however that protocol nets are strictly constructive: They can only add elements and places, it is not possible to remove parts of a net with them.

The more generalized and fluid nature of protocol nets means it is harder to prove algebraic properties over them. Protocol nets should be seen as a practical tool for designing systems with varied and plentiful interactions between numerous components, not as a way of proving mathematical truths over theoretical problems such as the Dining Philosophers. As we showed in the case study above, protocols are best used for 'black box' components that interact in all kinds of ways, where it might not be obvious from the start which component might be linked with what other component(s), and what type of interactions they will use.

The original inspiration for this project, and an application I still think protocol nets can be very useful for, is the modeling of biological systems. Biological systems tend to be hundreds or thousands of independent processes and pathways. Each process takes certain inputs, which can be various things: Molecules, strands of DNA to be transcribed, nerve signals, etc, and creates output, such as energy, other molecules and proteins, signals to different nerves, etc. Individual processes tend to be fairly simple and can be described using a Petri net. These processes often interact through these inputs and outputs, or by enabling or inhibiting other processes. We have shown that protocol nets can be used for to model this.

A similar case exists in economics: Modeling the interactions of thousands of different companies and other entities within an economic system that interact in all kinds of ways, such as by signing contracts, buying goods, or offering loans.

Given a set of protocol labels and nets, and an input of model nets that want to interact, the process of linking those nets and thus creating an overarching model could be done algorithmically, especially when involving hundreds or thousands of nets. Software could be written to facilitate this and easily allow people to create libraries of protocol nets.

In conclusion, we have created a tool that we hope is useful, especially to non-computer scientists who use Petri nets for modeling, such as biologists and economists.

## References


1. Farhad Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329âĂŞ366, 2004.
2. Eike Best, Raymond Devillers, and Maciej Koutny. *Petri Net Algebra*. Springer-Verlag, Berlin, Heidelberg, 2001.
3. Yorim Christiaanse. Project description of the b. subtilis petri net model. Internal LIACS report. Available on request, 2017.
4. Dorsaf Elhog-Benzina, Serge Haddad, and Rolf Hennicker. *Refinement and Asynchronous Composition of Modal Petri Nets*, pages 96–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
5. Rachid Hamadi and Boualem Benatallah. A petri net-based model for web service composition. In *ADC*, 2003.
6. Ekkart Kindler. A compositional partial order semantics for petri net components. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets 1997*, pages 235–252, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
7. Pieter M. Kwantes and Jetty Kleijn. On the synthesis of industry level process models from enterprise level process models. In Wil M. P. van der Aalst, Robin Bergenthum, and Josep Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2018 Satellite event of the conferences: 39th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2018 and 18th International Conference on Application of Concurrency to System Design ACSD 2018, Bratislava, Slovakia, June 25, 2018*, volume 2115 of *CEUR Workshop Proceedings*, pages 6–22. CEUR-WS.org, 2018.
8. Ross Overbeek, Ronald C Taylo, Neal Conrad, Veronika Vonstein, Vincent Fromion, Miguel Rocha, and Isabel Rocha. Reconstruction of the regulatory network for bacillus subtilis and reconciliation with gene expression data. *Frontiers in Microbiology (7), 11 p..(2016)*, 2016.
9. Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, UniversitÃďt Hamburg, 1962.
10. Wolfgang Reisig. Deterministic buffer synchronization of sequential processes. *Acta Informatica*, 18(2):117–134, Nov 1982.
11. Wolfgang Reisig. Simple composition of nets. In Giuliana Franceschinis and Karsten Wolf, editors, *Applications and Theory of Petri Nets*, pages 23–42, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
12. Wolfgang Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer Publishing Company, Incorporated, 2013.
13. Wolfgang Reisig. Associative composition of components with double-sided interfaces. *Acta Informatica*, 56(3):229–253, Apr 2019.
14. Grzegorz Rozenberg. Handbook of graph grammars and computing by graph transformation. 01 1997.
15. Vitali Schneider and Walter Vogler. Modal open petri nets. In Susanna Donatelli and Stefan Haar, editors, *Application and Theory of Petri Nets and Concurrency*, pages 25–46, Cham, 2019. Springer International Publishing.
16. Stefan Schuster, Thomas Pfeiffer, Ferdinand Moldenhauer, Ina Koch, and Thomas Dandekar. Exploring the pathway structure of metabolism: Decomposition into subnetworks and application to mycoplasma pneumoniae. *Bioinformatics (Oxford, England)*, 18:351–61, 03 2002.
17. Younes Souissi. On liveness preservation by composition of nets via a set of places. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1991*, pages 277–295, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.