

Ring Protocol

Abstract

This is a small toy example which is well-suited as a first introduction to hierarchical CP-nets. The use of substitution transitions and fusion sets are described in great detail.

The CPN model describes how a number of different sites communicate via a ring network.

The example is taken from Sect. 3.1 of Vol. 1 of the CPN book.

Developed and Maintained by:

Kurt Jensen, Aarhus University, Denmark (kjensen@daimi.aau.dk).

Graphical Quality

The figures in this document are inserted via PICT format. This is why some of the arcs and place borders look a bit ragged. A postscript printout from Design/CPN (and the screen image in Design/CPN) has much higher graphical quality.

CPN Model

The basic idea behind hierarchical CP-nets is to allow the modeller to construct a large model by combining a number of small CP-nets into a larger net. This is similar to the situation in which a programmer constructs a large program from a set of modules and subroutines.

Substitution of transitions

The intuitive idea behind substitution transitions is to allow the user to relate a transition (and its surrounding arcs) to a more complex CP-net – which usually gives a more precise and detailed description of the activity represented by the substitution transition. The idea is analogous to the hierarchy constructs found in many graphical description languages (e.g., data flow diagrams) and it is also, in some respects, analogous to the module concepts found in many modern programming languages. At one level, we want to give a simple description of the modelled activity without having to consider internal details about how it is carried out. At another level, we want to specify the more detailed behaviour. Moreover, we want to be able to integrate the detailed specification with the more crude description – and this integration must be done in such a way that it becomes meaningful to speak about the behaviour of the *combined* system. Now let us consider a small example, consisting of a simple ring network with four different sites. The main purpose of the example is to explain the semantics of substitution transitions, and thus the described network is far too simple to be realistic.

As mentioned above, we want to relate individual CP-nets to nodes, which are members of other CP-nets, and this means that our description will contain a *set* of non-hierarchical CP-nets – which we shall call **pages**. Below you see a page from the network system. The page has a **page name** *NetWork* (which is a text string) and a **page number** *10* (which is a non-negative integer).

The page *NetWork#10* contains ring consisting of four places and four transitions. The four transitions are **substitution transitions**. This can be seen because each of them has a small HS-tag adjacent to it (HS = Hierarchy + Substitution). The text strings in the dashed boxes next to the HS-tags are called **hierarchy inscriptions** and they define the details of the substitutions. A double-click on the HS-tag makes the hierarchy inscription (which may be quite large) visible/invisible. In this way it is possible to have a CP-net with a complex hierarchical structure without getting the individual pages too overloaded.

The first line of each hierarchy inscription tells us the identity of the **subpage**, i.e., the page which contains the detailed description of the activity modelled by the corresponding substitution transition. Each substitution transition is said to be a **superpage** (of the corresponding subpage) while the page of a substitution transition is a **subpage** (of the corresponding subpage). In our example, we can see that all four substitution transitions of *NetWork#10* share the same subpage *Site#11* which is shown below. This means that the hierarchical net will have four instances of *Site#11*. Each of these **page instances** will have its own private marking, which is

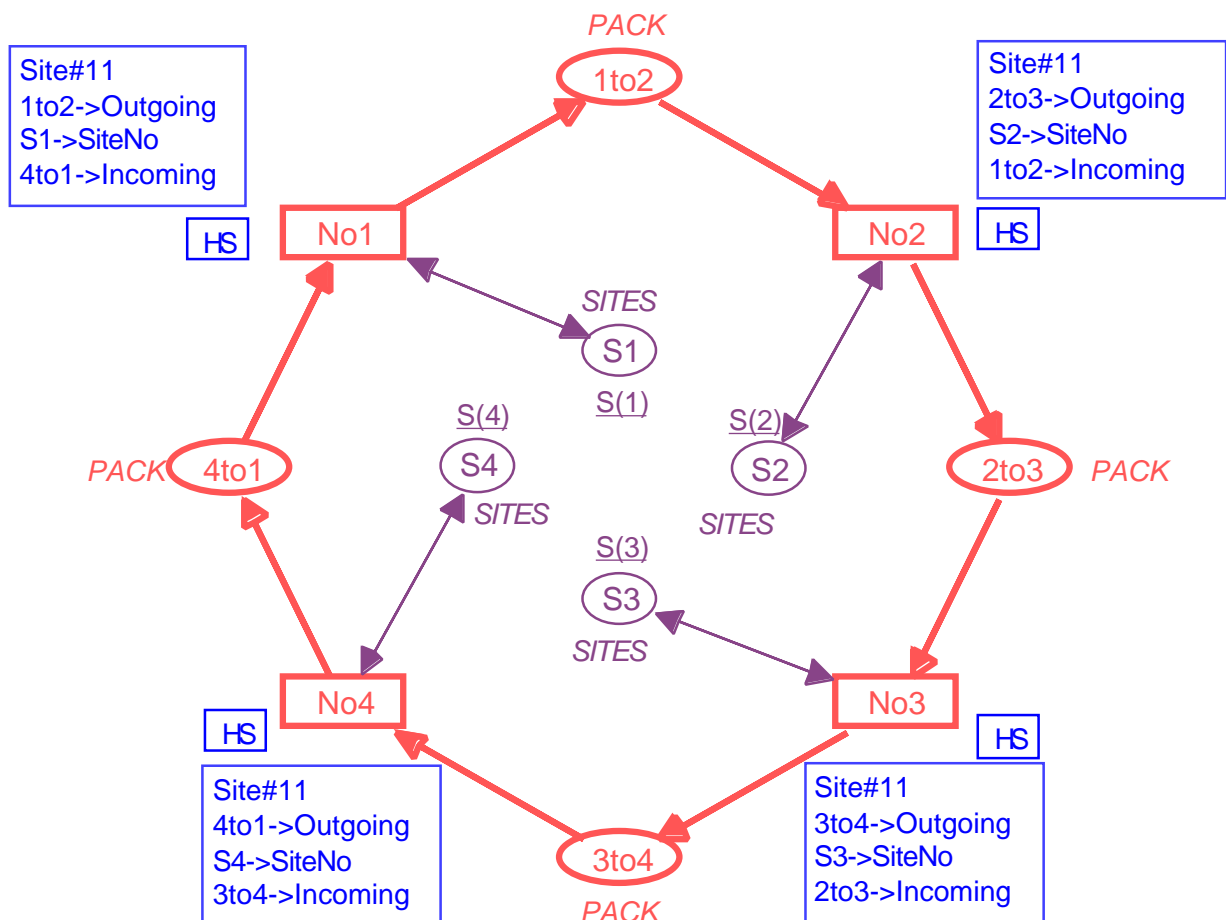
independent of the markings of the other instances (in a similar way that each procedure call has its own private copies of the local variables in the procedure).

When a CP-net is simulated by means of the CPN simulator, we have a window for each page. The window shows the marking of one page instance at a time, and it is possible for the user to switch from one instance to another. We will return to the remaining lines of the hierarchy inscriptions in a moment, but let us first take a look at *Site#11* which describes an individual site in the ring network.

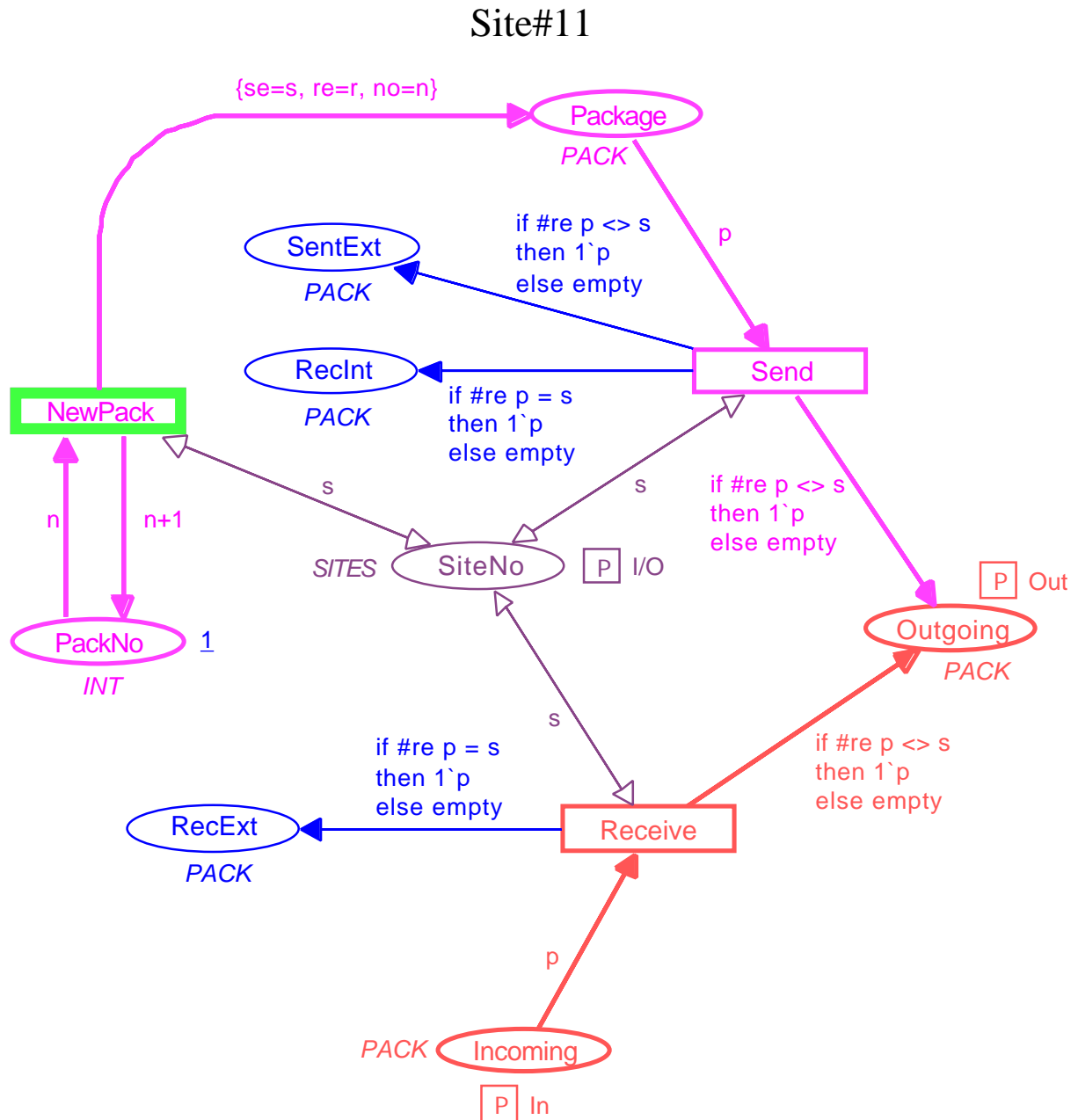
The declarations (at the bottom of the page) declares a constant $NoOfSites = 4$, which tells us how many sites the ring network has. Moreover, three colour sets are declared: INT contains all integers. SITES contains the elements $\{S(1), S(2), \dots, S(NoOfSites)\}$, which are used to identify the individual sites of the network. Finally, PACK describes the individual packages which are sent in the network. The format is a record, containing an se-field for the identity of the sender, an re-field for the identity of the receiver and a no-field for a package number. The actual data content of the packages is ignored, but if desired it could of course easily be added as an additional field of the record.

Site#11 has three different transitions. Each occurrence of *NewPack* creates a new package. The no-field of the new package is determined by the token on the place *PackNo* (and the colour of this token is increased by one so that the next package will get a package number which is one higher). The se-field of the new package is determined by the tokens at place *SiteNo*. Finally, the re-field is determined by

Network#10



the variable r – which does not appear in the guard and input arc expressions. This means that the binding of r does not influence the enabling, and thus r can take an arbitrary value (from $SITES$). This means that a site can send a package to itself, but if desired this could of course easily be prevented by adding a guard specifying that $r \neq s$.



```

val NoOfSites = 4;
color INT = int;
color SITES = index S with 1..NoOfSites declare ms;
color PACK = record se:SITES * re:SITES * no:INT;
var s,r : SITES;
var p : PACK;
var n : INT;

```

The created packages are handled by the transition *Send*, which inspects the re-field of the package. This is done by means of the expression $\#re\ p$ which denotes the re-field of the record p . When the re-field indicates that the receiver is different from the present site, the package is transferred to the place *Outgoing* (which is the “output gate” to the rest of the network) and a copy of the package is put on the place *SentExt* (indicating that the package is sent to an external receiver). Otherwise the package is sent directly to the place *RecInt* (indicating that the package is received from an internal sender).

Finally, the transition *Receive* inspects all the packages which arrive at the place *Incoming* (which is the “input gate” from the rest of the network). Again the re-field is inspected, and based on this inspection the package is routed, either to *Outgoing* or to *RecExt* (indicating that the package is received from an external sender).

The basic idea behind substitution transitions is that it should be possible to translate a hierarchical CP-net into a behaviourally equivalent non-hierarchical net by replacing each substitution node (and its surrounding arcs) with a copy of its subpage. However, to do this we need to know how the subpage should be “glued together” with the surroundings of the supernode (i.e., the rest of the superpage). This information is provided by the **port assignment**, which describes the interface between the superpage (*NetWork#10*) and the subpage (*Site#11*). The port assignment is contained in the remaining lines of the hierarchy inscriptions. Each such line relates a **socket node** on the superpage (i.e., one of the places that surrounds the substitution transition) with a **port node** on the subpage (i.e., one of the places which have an In-tag, Out-tag or I/O-tag next to it). An In-tag indicates that the port node must be related to a socket node which is an input node of the substitution transition (and it is not allowed also to be an output node). Analogously, an Out-tag indicates that the port must be related to a socket which is an output node (and not an input node), while an I/O-tag indicates that the socket must be both an input and output node.

In our example, let us consider the hierarchy inscription of transition *No1* (which represents site number one). The first line of the port assignment tells us that the socket node *Ito2* is assigned to (i.e., “glued” together with) the port node *Outgoing*. Intuitively, this means that the two places represent a *single* conceptual place – and thus they will always have identical markings. If the occurrence of a step adds or removes one or more tokens at one of the two places an identical set of tokens will be added/removed at the other. Analogously, the second and third lines tell us that the socket nodes *S1* and *4to1* are assigned to the port nodes *SiteNo* and *Incoming*, respectively. The remaining three hierarchy inscriptions (of *No2*, *No3* and *No4*) are interpreted in a similar way.

Page instances

In the ring network we have assumed that there is only a single page instance of *NetWork#10*, and that the only page instances of *Site#11* are the four which exist because of the four substitution transitions on *NetWork#10*. This is a rather obvious choice – as long as we want to model a single ring with four individual sites.

However, what if we instead want to model a larger system containing two totally independent ring networks? Then we would want *NetWork#10* to have two page instances and *Site#11* to have eight page instances (four for each ring). This discussion indicates that we need a way of specifying the number of page instances which a system has. The situation is analogous to a program, where we also need to specify a main program, or a set of start objects which then invoke all the other procedures and objects of the program.

To specify the number of page instances we take a very general and simple strategy. The basic idea is that the user specifies a set of **prime page instances** (which are analogous to the main program or the start objects). As indicated above we may want the same page to have two or more prime page instances – and this means that the prime page instances are specified as a *multi-set* of pages. Those pages which are members of the multi-set (i.e., have a non-zero coefficient) are called **prime pages**. In many cases, e.g., the ring network, we will have a multi-set of prime page instances which contains only a single element (with coefficient one).

In addition to the prime page instances the system will have a number of **secondary page instances** (which are analogous to the procedures and objects which are invoked by the main program or the start objects). Each substitution transition on a prime page instance will generate a secondary page instance, and if any of these secondary page instances have substitution transitions these will generate secondary page instances as well, and so on. In the ring network the single prime page instance (of *NetWork#10*) generates four secondary page instances (of *Site#11*).

To give an overview of the set of pages, the multi-set of prime and secondary page instances, and the superpage/subpage relationships in a hierarchical CP-net, we use a **page hierarchy graph**. This is a directed graph which contains a node for each page and an arc for each direct superpage/subpage relationship. Each node is inscribed by the name and number of the corresponding page, while each arc is inscribed with the names of the corresponding substitution transitions (i.e., the names of those substitution transitions which belong to the page of the source node, and have the page of the destination node as direct subpage). The set of prime page instances is indicated by positioning the word “Prime” next to each prime page, adding a number, e.g., “Prime : 3” if the coefficient differs from one. The page hierarchy graph for the ring network is shown below. Notice that the arc has four inscriptions (because *NetWork#10* has four substitution transitions).



The page hierarchy graph can be used to find the number of page instances of a given page. This can be calculated as the number of different “routes” which leads from an appearance of a prime page instance to the page in question (when an arc represents more than one substitution transition there are routes for each of these). Routes of length zero represent prime page instances, while all other routes represent secondary page instances.

Fusion of places

The intuitive idea behind fusion of places is to allow the user to specify that a set of places are considered to be identical, i.e., they all represent a single conceptual place even though they are drawn as individual places. This means that when a token is added/removed at one of the places, an identical token will be added/removed at all the others. The relationship between the members of a fusion set is, in many respects, similar to the relationship between two places which are assigned to each other by a port assignment.

The places that participate in such a **fusion set** may belong to a single page or to several different pages. When all members of a fusion set belong to a single page and that page only has one page instance, place fusion is nothing other than a drawing convenience that allows the user to avoid too many crossing arcs. However, things become much more interesting when the members of a fusion set belong to several different pages *or* to a page that has several page instances. In that case fusion sets allow the user to specify a behaviour which it may be cumbersome to describe without fusion.

There are three different kinds of fusion sets: **global fusion sets** are allowed to have members from many different pages, while **page fusion sets** and **instance fusion sets** only have members from a single page. The difference between the last two is the following. A page fusion unifies all the instances of its places (independently of the page instance at which the place instance appear), and this means that the fusion set only has one “resulting place” which is “shared” by all instances of the corresponding page. In contrast, an instance fusion set only identifies place instances that belong to the *same* page instance, and this means that the fusion set has a “resulting place” for each page instance. The semantics of a global fusion set is analogous to that of a page fusion set – in the sense that there only is one “resulting place” (which is common for all instances of all the participating pages). To allow modular analysis of hierarchical CP-nets, global fusion sets should be used with care.

To investigate fusion sets you may use Design/CPN to modify the “Ring Network” by defining two fusion sets on *Site#11*: an instance fusion set containing the two places *RecExt* and *RecInt*, and a page fusion set containing a single place *SentExt*. The first fusion set combines *RecExt* and *RecInt* into a single place – and since we deal with an instance fusion set, we get a “resulting place” for each of the four page instances of *Site#11*. The second fusion set combines *SentExt* with itself – but now we deal with a page fusion set, and thus there is only one “resulting place” shared by all four page instances.