Modeling protocol behaviour in π-calculus using the Mobility Workbench [*]

Karl Palmskog
(palmskog@kth.se)

Abstract

We model the behaviour of a mobility-enhanced network protocol stack in the pure π-calculus, and use the Mobility Workbench (MWB) tool to analyze various properties of the model. In this way, we explore the feasibility of using π and MWB in verifying the Session Management Protocol, which resides on top of TCP/IP and provides generic session management with mobility for applications.

1 Introduction

The π-calculus was proposed as a mathematical framework for specifying and analyzing mobile processes by Milner et al [3] in 1989. Judging by the quantity and quality of the work in the area of mobile process calculi since then, it may well be said that π-calculus has become the established way of formally describing parallelism, interaction and mobility in computing [4]. In π-calculus, processes synchronize and exchange names; these names may be used as links for transmission of other names, allowing dynamic process reconfiguration. As Milner says [2, p. 78], the mobility of a process is determined by the links available to it.

One important real-world mobility problem, which has grown in parallel with the π-calculus, is the development of a protocol stack capable of handling mobility of devices connected to a network — see e.g. [9] for an overview. There is no consensus which protocol layer should be responsible for handling mobility; one approach favours the network layer and has resulted in the Mobile IP protocol [7]. Mobile IP introduces a proxy with invariant address between a mobile device and its peers, and thus does not provide true end-to-end mobility. Another more recent approach to the problem, which does not make use of proxies, is described in [1]. The idea is to make end-to-end connections suspendable and resumeable at will by introducing new session-layer functionality.

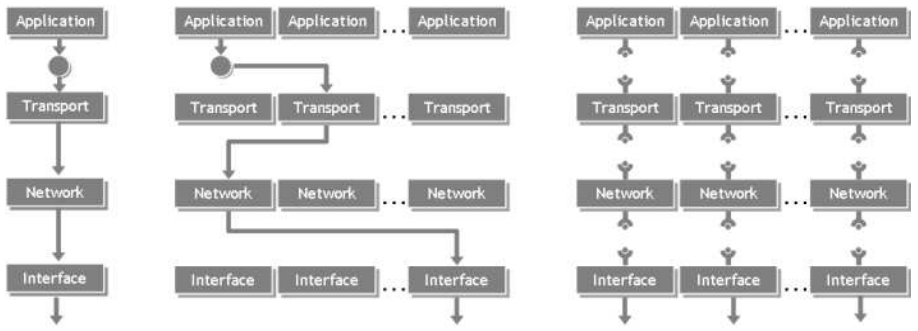
Using the Mobility Workbench [12], a mature tool for the verification of π-calculus processes, we model and examine the behaviour of a network protocol stack which has been augmented with the link mobility required for session persistence, as was done in the Linux kernel in [13]. We aim to explore the feasibility, and highlight the challanges, in using π and the MWB in such an endeavor, in the context of a Master's Thesis project at Ericsson Research for the verification of the Session Management Protocol [14].

[*]Originally a project report in the course 2G1517 "Advanced Formal Metods" at KTH.

2 A mobility-enhanced network stack

In a traditional interface to the protocol stack, every connection must stick to the initial choice of layer components (e.g. wired/IPv4/TCP) throughout its lifetime, as is depicted in the leftmost column in Figure 1 below. An unexpected event anywhere in the stack will break the current connection and require the establishment of a new, unrelated connection [13, p. 10]. To have a truly persistent session, we must be able to switch on-the-fly between different protocols at the transport layer (TCP/UDP/SCTP), different protocols at the network layer (IPv4/IPv6), and different transmission mechanisms at the physical layer (wired/wireless).

Figure 1: Traditional versus mobile network stacks (from [13]).



We may view the protocol stack as consisting of a matrix of protocol components, where each component may have links to components in the layer below and in the layer above. The rightmost column in Figure 1 illustrates this. The matrix lends itself well for expressing in the π-calculus, since π is all about movement of links in virtual link space, and such movement is conceptually what takes place. Finally, we expect there to be at most one active link between each layer, but we do not expect there to always be a path through the stack as in the center column in the figure.

3 Modeling, analysis and results

Before we model the actual stack, we turn to the problem of finding a simple specification that our model must conform to if it is correct. In doing this, we follow the outline provided by Parrow in [6]. Parrow expresses the layer service specification in CCS as

$$ServiceSpec \triangleq sendmessage(m).receivemessage(m).ServiceSpec$$

The idea is that each layer should behave from the outside as a perfect buffer. Using a specification similar to this one, we may test each layer implementation separately or several at a time. We say that a layer implementation conforms to its specification in π-calculus when we can find exhibit a bisimulation according to the theory of Sangiorgi [8]. The π-calculus variation of $ServiceSpec$ which we provide is as follows:

$$Sender(link, datum) \triangleq \overline{datum}.\overline{link}\langle datum\rangle.link(x).[x = datum]datum.Sender(link, datum)$$

$$Receiver(link) \triangleq link(x).\overline{link}\langle x\rangle.Receiver(link)$$

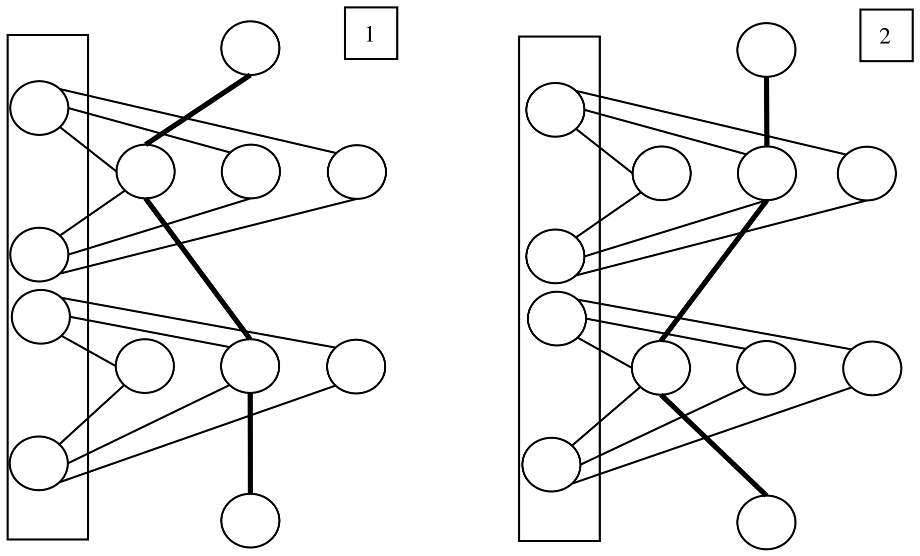
$$Spec(datum) \triangleq \text{new}\, link(Sender(link, datum) \mid Receiver(link))$$

In this specification, we transmit the parameter *datum*, and expect this name to later be returned to our sender process for retransmission. In an incorrect π representation of a protocol stack, the message may get permanently stuck somewhere, or the wrong message may be returned. Such a representation will not be bisimilar to $Spec(datum)$.

Returning to our problem of modeling the protocol stack matrix, we start by considering the protocol components, or nodes, at each layer. An active component must obviously have two links: one connecting it to the layer above, and one connecting it to the layer below. However, if we expect it to be able to lose these links to another node on the same layer, the active node must have two additional links — one for transmitting its upper link and one for transmitting its lower link. This leads to nodes having four principal states: one state for when it possesses both links, two for when it possesses the one or the other, and one when it has no links. To facilitate the transmission of message links, we let there be controller processes, which indeterministically receive links and transmit them to other nodes in the same layer.

Considering a stack with only two layers, we can represent two configurations of our π model graphically as in Figure 2 below. To separate the controller processes from the network nodes, the former are enclosed in a rectangle. Notice that a reconfiguration of the model from (1) to (2) would involve many commitments, since no less than four endpoints of links are different.

Figure 2: Graphical representation of the π model and its configurations.



We are now ready to present an initial one-layer model, with three nodes, in the syntax of [2]. It has only one-way communication, and hence the bottom, physical-layer process transmits the datum directly back to the application-level process. Overall, the model is somewhat similar to the mobile network of [5] as seen in [2, pp. 80–83].

$$\begin{aligned}
App(send, recv, datum) &\triangleq \overline{datum}.\overline{send}\langle datum\rangle.recv(x).[x = datum]datum.App(send, recv, datum)\\
Phy(send, recv) &\triangleq recv(x).\overline{send}\langle x\rangle.Phy(send, recv)\\
Spec(datum) &\triangleq \text{new}\, ln1, ln2\,(App(ln1, ln2, datum) \mid Phy(ln2, ln1))
\end{aligned}$$

$$\begin{aligned}
ProtNodeActive(rcontrol, scontrol, recv, send) &\triangleq recv(x).\overline{send}\langle x\rangle.\\
&\quad ProtNodeActive(rcontrol, scontrol, recv, send)\\
&\quad + \overline{rcontrol}\langle recv\rangle.ProtNodeSend(rcontrol, scontrol, send)\\
&\quad + \overline{scontrol}\langle send\rangle.ProtNodeRecv(rcontrol, scontrol, recv)\\
ProtNodeSend(rcontrol, scontrol, send) &\triangleq rcontrol(recv).\\
&\quad ProtNodeActive(rcontrol, scontrol, recv, send)\\
&\quad + \overline{scontrol}\langle send\rangle.ProtNodeInactive(rcontrol, scontrol)\\
ProtNodeRecv(rcontrol, scontrol, recv) &\triangleq scontrol(send).\\
&\quad ProtNodeActive(rcontrol, scontrol, recv, send)\\
&\quad + \overline{rcontrol}\langle recv\rangle.ProtNodeInactive(rcontrol, scontrol)\\
ProtNodeInactive(rcontrol, scontrol) &\triangleq rcontrol(recv).\\
&\quad ProtNodeRecv(rcontrol, scontrol, recv)\\
&\quad + scontrol(send).ProtNodeSend(rcontrol, scontrol, send)
\end{aligned}$$

$$\begin{aligned}
Control(control1, control2, control3) &\triangleq control1(ln).\\
&\quad (\overline{control2}\langle ln\rangle.Control(control2, control1, control3)\\
&\quad + \overline{control3}\langle ln\rangle.Control(control3, control2, control1))
\end{aligned}$$

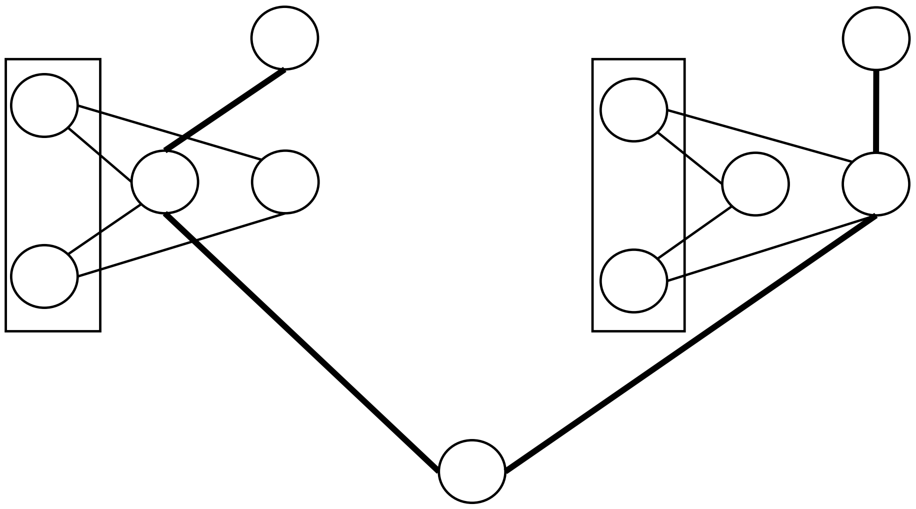
$$\begin{aligned}
System(datum) \triangleq\ &\text{new}\, ucontrol1, ucontrol2, ucontrol3,\\
&dcontrol1, dcontrol2, dcontrol3, ln1, ln2, ln3\\
&\big(App(ln1, ln3, datum) \mid Phy(ln3, ln2) \mid Control(ucontrol1, ucontrol2, ucontrol3) \mid\\
&Control(dcontrol1, dcontrol2, dcontrol3) \mid ProtNodeActive(ucontrol1, dcontrol1, ln1, ln2) \mid\\
&ProtNodeInactive(ucontrol2, dcontrol2) \mid ProtNodeInactive(ucontrol3, dcontrol3)\big)
\end{aligned}$$

As the Mobility Workbench confirms (after considerable time), it is indeed the case that

$$Spec(datum) \approx System(datum).$$

Next, we try to express a system where we have two-way communication, and where there are two application nodes communication through two stacks — a reasonable real-world scenario. A graphical view of such a system can be seen in Figure 3 on the next page.

Figure 3: A two-stack, two-way communication model.



The syntax for the model is as follows:

$$AppSender(link, datum) \triangleq \overline{datum}.\overline{link}\langle datum\rangle.link(x).[x = datum]datum.AppSender(link, datum)$$

$$AppReceiver(link) \triangleq link(x).\overline{link}\langle x\rangle.AppReceiver(link)$$

$$Spec(datum) \triangleq ({}^{l}ink)(AppSender(link, datum)|AppReceiver(link))$$

$$Phy(llink, rlink) \triangleq llink(x).\overline{rlink}\langle x\rangle.Phy(llink, rlink) + rlink(x).\overline{llink}\langle x\rangle.Phy(llink, rlink)$$

$$\begin{aligned}ProtNodeActive(ucontrol, dcontrol, ulink, dlink) &\triangleq ulink(x).\overline{dlink}\langle x\rangle.\\ &ProtNodeActive(ucontrol, dcontrol, ulink, dlink) + dlink(x).\overline{ulink}\langle x\rangle.\\ &ProtNodeActive(ucontrol, dcontrol, ulink, dlink) + \overline{ucontrol}\langle ulink\rangle.\\ &ProtNodeOnlyDown(ucontrol, dcontrol, dlink) + \overline{dcontrol}\langle dlink\rangle.\\ &ProtNodeOnlyUp(ucontrol, dcontrol, ulink)\end{aligned}$$

$$\begin{aligned}ProtNodeOnlyDown(ucontrol, dcontrol, dlink) &\triangleq ucontrol(ulink).\\ &ProtNodeActive(ucontrol, dcontrol, ulink, dlink) + \overline{dcontrol}\langle dlink\rangle.\\ &ProtNodeInactive(ucontrol, dcontrol)\end{aligned}$$

$$\begin{aligned}ProtNodeOnlyUp(ucontrol, dcontrol, ulink) &\triangleq dcontrol(dlink).\\ &ProtNodeActive(ucontrol, dcontrol, ulink, dlink) + \overline{ucontrol}\langle ulink\rangle.\\ &ProtNodeInactive(ucontrol, dcontrol)\end{aligned}$$

$$\begin{aligned}ProtNodeInactive(ucontrol, dcontrol) &\triangleq dcontrol(dlink).\\ &ProtNodeOnlyDown(ucontrol, dcontrol, dlink) + ucontrol(ulink).\\ &ProtNodeOnlyUp(ucontrol, dcontrol, ulink)\end{aligned}$$

$$
\begin{aligned}
&\mathit{System}(\mathit{datum}) \triangleq \text{new}\, \mathit{ucontrol}11, \mathit{ucontrol}12, \mathit{ucontrol}21, \mathit{ucontrol}22,\\
&\quad \mathit{dcontrol}11, \mathit{dcontrol}12, \mathit{dcontrol}21, \mathit{dcontrol}22, \mathit{link}1, \mathit{link}2, \mathit{link}3, \mathit{link}4\\
&\quad \big(\mathit{AppSender}(\mathit{link}1, \mathit{datum}) \mid \mathit{Control}(\mathit{ucontrol}11, \mathit{ucontrol}12) \mid\\
&\qquad \mathit{Control}(\mathit{dcontrol}11, \mathit{dcontrol}12) \mid \mathit{Control}(\mathit{ucontrol}21, \mathit{ucontrol}22) \mid\\
&\qquad \mathit{Control}(\mathit{dcontrol}21, \mathit{dcontrol}22) \mid \mathit{ProtNodeActive}(\mathit{ucontrol}11, \mathit{dcontrol}11, \mathit{link}1, \mathit{link}2) \mid\\
&\quad \mathit{ProtNodeInactive}(\mathit{ucontrol}12, \mathit{dcontrol}12) | \mathit{Phy}(\mathit{link}2, \mathit{link}3) \mid\\
&\quad \mathit{ProtNodeActive}(\mathit{ucontrol}21, \mathit{dcontrol}21, \mathit{link}4, \mathit{link}3) \mid\\
&\quad \mathit{ProtNodeInactive}(\mathit{ucontrol}22, \mathit{dcontrol}22) | \mathit{AppReceiver}(\mathit{link}4)\big)
\end{aligned}
$$

This time, the Mobility Workbench was unable to answer in reasonable time whether $\mathit{Spec}(\mathit{datum}) \approx \mathit{System}(\mathit{datum})$. However, using the `step` function, the behaviour of the model seemed to be as intended. Also, a bisimulation was successfully established when the following more simple model was used:

$$
\begin{aligned}
&\mathit{System}'(\mathit{datum}) \triangleq \text{new}\, \mathit{ucontrol}11, \mathit{ucontrol}12, \mathit{dcontrol}11,\\
&\quad \mathit{dcontrol}12, \mathit{link}1, \mathit{link}2, \mathit{link}3\\
&\quad \big(\mathit{AppSender}(\mathit{link}1, \mathit{datum}) \mid \mathit{Control}(\mathit{ucontrol}11, \mathit{ucontrol}12) \mid\\
&\quad \mathit{Control}(\mathit{dcontrol}11, \mathit{dcontrol}12) \mid \mathit{ProtNodeActive}(\mathit{ucontrol}11, \mathit{dcontrol}11, \mathit{link}1, \mathit{link}2) \mid\\
&\quad \mathit{ProtNodeInactive}(\mathit{ucontrol}12, \mathit{dcontrol}12) \mid \mathit{Phy}(\mathit{link}2, \mathit{link}3) \mid \mathit{AppReceiver}(\mathit{link}3)\big)
\end{aligned}
$$

4 Conclusions and further work

As was suspected, expressing a mobility-enhanced network protocol stack in π-calculus turned out to be straightforward. However, the inefficiency of the Mobility Workbench `weq` command (which checks for bisimulation between processes) turned out to be a problem. One solution is to only use a small number of processes with few states. Another solution, described in [11], is to translate π processes into the PROMELA process langauge, and use the SPIN model checker for verification. However, while SPIN is a vastly more efficient tool than the Mobility Workbench [11, p. 13], the translation of processes into PROMELA and of modal logic specification formulae into linear time logic formulae is not trivial and must largely be done manually.

The investigation has focused on modeling and bisimulation equivalence, but extended formal system specification in modal logics, e.g. in the modal π-μ-calculus, is certainly within the scope of the Master's Thesis and is thus being worked on. Also, there are some open questions pertaining to the presented π protocol stack models. As is readily seen, the protocol nodes must be specified manually, whether they are active or not. Ideally, the system should be able to generate new nodes at will, in the style of the elastic buffer in [2, pp. 148*ff*]. Furthermore, it is not unreasonable to let inactive nodes be (indeterministically) garbage-collected.

An attentive reader may have noticed that messages can get stuck in the protocol stack if a link is moved at an unappropriate time. In an implementation, such an event may cause a message to get lost and consequently re-sent. However, in the present model, a message is stalled until the link returns. In some contexts, it may be necessary to model message losses explicitly, but here we have not found that this is the case.

References

[1] B. Landfeldt, T. Larsson, Y. Ismailov, and A. Seneviratne. SLM, a framework for session layer mobility management. *ICCCN99*, Oct. 1999.

[2] R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, May 1999.

[3] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, Sept. 1992. http://www.lfcs.informatics.ed.ac.uk/reports/89/ECS-LFCS-89-85/.

[4] U. Nestmann and B. Victor. Calculi for mobile processes: Bibliography and web pages. *Bulletin of the EATCS*, 64:139–144, Feb. 1998.

[5] F. Orava and J. Parrow. An algebraic verification of a mobile network. *Journal of Formal Aspects of Computing*, 4:497–543, 1992. http://citeseer.ist.psu.edu/orava91algebraic.html.

[6] J. Parrow. Verifying a CSMA/CD-protocol with CCS. In *Proceedings of the IFIP WG6.1 Eighth International Symposium on Protocol Specification, Testing, and Verification*, pages 373–384, June 1988. http://www.imit.kth.se/courses/2G1516/Docs05/CCS/CSMA-CD-Parrow.pdf.

[7] C. Perkins. RFC 2002: IP mobility support, Oct. 1996. Updated by RFC2290 [10]. Status: PROPOSED STANDARD.

[8] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996. An extract appeared in *Proc. CONCUR '93*, Lecture Notes in Computer Science 715, Springer Verlag.

[9] A. Snoeren, H. Balakrishnan, and F. Kaashoek. Reconsidering Internet Mobility. In *8th Workshop on Hot Topics in Operating Systems*, Elmau/Oberbayern, Germany, May 2001.

[10] J. Solomon and S. Glass. RFC 2290: Mobile-IPv4 configuration option for PPP IPCP, Feb. 1998. Updates RFC2002 [7]. Status: PROPOSED STANDARD.

[11] H. Song and K. J. Compton. Verifying π-calculus processes by Promela translation. *Technical Report 472, University of Michigan*, 2003. http://www.eecs.umich.edu/techreports/cse/03/CSE-TR-472-03.pdf.

[12] B. Victor. *A Verification Tool for the Polyadic π-Calculus.* Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50. http://www.docs.uu.se/~victor/tr/docs-tr-94-50.html.

[13] Y. Wang. Mobility support for networked applications built in the TCP/IP stack. Master's thesis, KTH, 2006.

[14] M. Widell and P. Arvidsson. Design of the session layer supporting mobile, delay and disconnection tolerant communication. Master's thesis, KTH, 2006. To appear.

A MWB source code

A.1 One-way communication, three-node model

```
agent App(send,recv,datum) = 'datum.'send<datum>.recv(x).[x=datum]
    datum.App(send,recv,datum)

agent Phy(send,recv) = recv(x).'send<x>.Phy(send,recv)

agent Spec(datum) = (^ln1,ln2) (App(ln1,ln2,datum) | Phy(ln2,ln1))

agent ProtNodeActive(rcontrol,scontrol,recv,send) = recv(x).'send<x>.
    ProtNodeActive(rcontrol,scontrol,recv,send) + 'rcontrol<recv>.
    ProtNodeSend(rcontrol,scontrol,send) + 'scontrol<send>.ProtNodeRecv(
    rcontrol,scontrol,recv)

agent ProtNodeSend(rcontrol,scontrol,send) = rcontrol(recv).ProtNodeActive(
    rcontrol,scontrol,recv,send) + 'scontrol<send>.ProtNodeInactive(rcontrol
    ,scontrol)

agent ProtNodeRecv(rcontrol,scontrol,recv) = scontrol(send).ProtNodeActive(
    rcontrol,scontrol,recv,send) + 'rcontrol<recv>.ProtNodeInactive(rcontrol,
    scontrol)

agent ProtNodeInactive(rcontrol,scontrol) = rcontrol(recv).ProtNodeRecv(
    rcontrol,scontrol,recv) + scontrol(send).ProtNodeSend(rcontrol,scontrol,
    send)

agent Control(control1, control2, control3) = control1(ln).('control2<ln>.
    Control(control2,control1,control3) + 'control3<ln>.Control(control3,
    control2,control1))

agent System(datum) = (^ucontrol1,ucontrol2,ucontrol3,dcontrol1,dcontrol2,
    dcontrol3,ln1,ln2,ln3) (App(ln1,ln3, datum) | Phy(ln3,ln2) | Control(
    ucontrol1,ucontrol2,ucontrol3) | Control(dcontrol1,dcontrol2,dcontrol3) |
    ProtNodeActive(ucontrol1, dcontrol1, ln1, ln2) | ProtNodeInactive(
    ucontrol2, dcontrol2) | ProtNodeInactive(ucontrol3, dcontrol3))
```

A.2 Two-way communication, two-stack model

```
agent AppSender(link,datum) = 'datum.'link<datum>.link(x).[x=datum]
      datum.AppSender(link,datum)

agent AppReceiver(link) = link(x).'link<x>.AppReceiver(link)

agent Spec(datum) = (^link) (AppSender(link,datum) | AppReceiver(link))

agent Phy(llink,rlink) = llink(x).'rlink<x>.Phy(llink,rlink) + rlink(x).'llink<x
      >.Phy(llink,rlink)

agent ProtNodeActive(ucontrol,dcontrol,ulink,dlink) = ulink(x).'dlink<x>.
      ProtNodeActive(ucontrol,dcontrol,ulink,dlink) + dlink(x).'ulink<x>.
      ProtNodeActive(ucontrol,dcontrol,ulink,dlink) + 'ucontrol<ulink>.
      ProtNodeOnlyDown(ucontrol,dcontrol,dlink) + 'dcontrol<dlink>.
      ProtNodeOnlyUp(ucontrol,dcontrol,ulink)

agent ProtNodeOnlyDown(ucontrol,dcontrol,dlink) = ucontrol(ulink).
      ProtNodeActive(ucontrol,dcontrol,ulink,dlink) + 'dcontrol<dlink>.
      ProtNodeInactive(ucontrol,dcontrol)

agent ProtNodeOnlyUp(ucontrol,dcontrol,ulink) = dcontrol(dlink).
      ProtNodeActive(ucontrol,dcontrol,ulink,dlink) + 'ucontrol<ulink>.
      ProtNodeInactive(ucontrol,dcontrol)

agent ProtNodeInactive(ucontrol,dcontrol) = dcontrol(dlink).
      ProtNodeOnlyDown(ucontrol,dcontrol,dlink) + ucontrol(ulink).
      ProtNodeOnlyUp(ucontrol,dcontrol,ulink)

agent Control(control1, control2) = control1(ln).'control2<ln>.control2(ln).'
      control1<ln>.Control(control1,control2)

agent System(datum) = (^ucontrol11,ucontrol12,ucontrol21,ucontrol22,
      dcontrol11,dcontrol12,dcontrol21,dcontrol22,link1,link2,link3,link4) (
      AppSender(link1,datum) | Control(ucontrol11,ucontrol12) | Control(
      dcontrol11,dcontrol12) | Control(ucontrol21,ucontrol22) | Control(
      dcontrol21,dcontrol22) | ProtNodeActive(ucontrol11,dcontrol11,link1,
      link2) | ProtNodeInactive(ucontrol12,dcontrol12) | Phy(link2,link3) |
      ProtNodeActive(ucontrol21,dcontrol21,link4,link3) | ProtNodeInactive(
      ucontrol22,dcontrol22) | AppReceiver(link4))

agent System'(datum) = (^ucontrol11,ucontrol12,dcontrol11,dcontrol12,link1,
      link2,link3) (AppSender(link1,datum) | Control(ucontrol11,ucontrol12) |
      Control(dcontrol11,dcontrol12) | ProtNodeActive(ucontrol11,dcontrol11,
      link1,link2) | ProtNodeInactive(ucontrol12,dcontrol12) | Phy(link2,link3)
       | AppReceiver(link3))
```