Examples for Multiparty Asynchronous Session Types

Marco Carbone[1,2] Kohei Honda[1] Nobuko Yoshida[2]

[1]Queen Mary, University of London, UK
[2] Imperial College, London, UK

1. Introduction

This manuscript contains examples of multiparty interactions for analysing the relationship between an approach based on global message flow and another centring on local (end-point) behaviours. Both approaches are based on a common feature: *structured representation of communications* (or *Session Types*). The global types originate from *the abstract version of* Choreography Description Language (CDL) [**6**], a web service description language developed by W3C's WS-CDL Working Group. The local calculus is based on the π-calculus [**4**] with multiparty session types, one of the representative calculi for communicating processes.

Both types are based on a notion of structured communication, called *session*. A session binds a series of communications between multi-parties into one, distinguishing them from communications belonging to other sessions. This is a standard practice in business protocols (where an instance of a protocol should be distinguished from another instance of the same or other protocols) and in distributed programming (where interacting parties use multiple TCP connections for performing a unit of conversation). As we shall explore in the present document, the notion of session can be cleanly integrated with such notions as branching, recursion (loop) and exceptions. We show, through examples taken from simple but non-trivial business protocols, how concise structured description of non-trivial interactive behaviour is possible using sessions. From a practical viewpoint, a session gives us the following merits.

- It offers a clean way to describe a complex sequence of communications with rigorous operational semantics, allowing structured description of interactive behaviour.
- Session-based programs can use a simple, algorithmically efficient typing algorithm to check its conformance to expected interaction structures.
- Sessions offer a high-level abstraction for communication behaviour upon which further refined reasoning techniques, including type/transition/logic-based ones, can be built.

In the remainder, we show two examples, one is basic one and another is larger one. These examples come from use-cases for CDL found in CDL primer [**5**] by Steve Ross-Talbot and Tony Fletcher, and those examples communicated by Gary Brown [**1**] and Nickolas Kavanztas [**3**].

We also use a slightly different syntax from the main paper for in order to gain the readability and follow CDL more directly. In particular, we make the participants explicit to give more intuitive ideas to relate global types and end-point representations. For a similar syntax, see [**2**] for the formal notation.

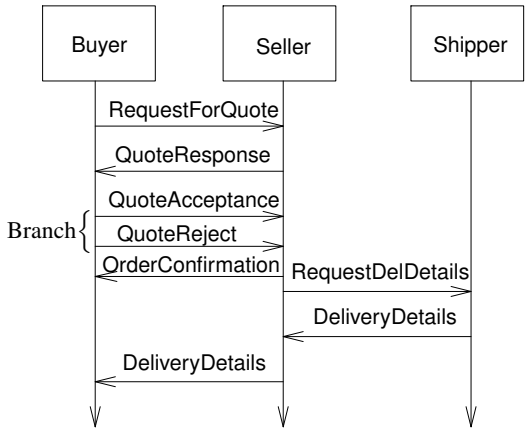

FIGURE 1. Graphical Representation of Simple Protocol

.

2. Example 1

2.1. A Business Protocol. We show how increasingly complex, business protocols can be accurately and concisely described in global types and the π-calculus with multiparty sessions, one based on global message flows and another based on local, or end-point, behaviours. Along the way we also illustrate each construct and types.

Our starting point is a simple business protocol for purchasing a good among a buyer, a seller and a shipper, which we call **BSH Protocol**. Informally the expected interaction is described as follows.

(1) First, Buyer asks Seller, through a specified channel, to offer a quote (we assume the good to buy is fixed).
(2) Then Seller replies with a quote.
(3) Buyer then answers with either QuoteAcceptance or QuoteRejection. If the answer is QuoteAcceptance, then Seller sends a confirmation to Buyer, and sends a channel of Buyer to Shipper. Then Shipper sends the delivery details to Buyer, and the protocol terminates. If the answer is QuoteRejection, then the interaction terminates.

Figure 1 presents an UML sequence diagram of this protocol. Observe that, in Figure 1, many details are left unspecified: in real interaction, we need to specify, for example, the types of messages and the information exchanged in interaction, etc.

2.2. Assumption on Underlying Communication Mechanisms. We first outline the basic assumptions common to both global and local formalisms. Below and henceforth we call the dramatis personae of a protocol (Buyer, Seller and Shipper in the present case), *participants*.

- In communication:
  (1) A sender participant sends a message and a receiver receives it, i.e. we only consider a point-to-point communication. A communication is always done through a *channel*. The message in a communication consists of an operator name and, when there is a value passing, a value. The value will be assigned to a local variable at the receiver's side upon the arrival of that message.

(2) Communication can be either an *in-session communication* which belongs to a session, or *session initiation channels* which establishes a session (which may be liked to establishing one or more fresh transport connections for a piece of conversation between multi distributed peers). In a session initiation communication, one or more fresh session channels belonging to a session are declared, i.e. one session can use multiple channels.
(3) A channel can be either a *session channel* which belongs to a specific session or an *session-initiating channel* which is used for session-initiation. For a session-initiating channel, we assume its sender and a receiver is pre-determined.

- We demand:
  (1) the order of messages from one participant to another through a specified channel is preserved.
  (2) one party participating in a session can use a session-channel both for sending and receiving.

As we discussed in the main paper, we can uniformly change these assumptions to pure asynchrony (by dropping the first condition) or to synchrony (by assuming a sender immediately knows the arrival of a message at a receiver). In this manuscript, we assume the same ordering in the main paper (i.e. message preserving order semantics).

2.3. Representing Communication (1): Initiating Session. Buyer's session-initiating communication in BSH Protocol is described in the global type as follows:[1]

$$\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch}) \,.\, G \tag{1}$$

which says:

> Buyer initiates a session with Seller by communication through a shared-initiating channel INITB2S, declaring a fresh in-session channel *B2Sch*. Then interaction moves to *G*.

Note "." indicates sequencing, as in process calculi. A session initiation can specify more than one session channels as needed, as the following example shows.

$$\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{InitB2S}(\mathit{B2Sch},\ \mathit{S2Bch}) \,.\, G \tag{2}$$

which declares two (fresh) session channels, one from Buyer to Seller and another in the reverse direction.

In local description, the behaviour is split into two, one for Buyer and another for Seller, using the familiar notation from process algebras. For example (1) becomes:

$$\mathsf{Buyer}[\ \mathsf{InitB2S}\,[2](\mathit{B2Sch}) \,.\, P_1\ ], \qquad \mathsf{Seller}[\ \overline{\mathsf{InitB2S}}\,[2](\mathit{B2Sch}) \,.\, P_2\ ] \tag{3}$$

Above $\mathsf{Buyer}[\ P\ ]$ specifies a buyer's behaviour, while $\mathsf{Seller}[\ P\ ]$ specifies a seller's behaviour. The over-lined channel indicates it is used for output (this follows the tradition of CCS/π-calculus: in CSP, the same action is written $\mathsf{InitB2S}\,!\,(\mathtt{B2Sch})$.

Note the behaviour of each participant is described rather than their interaction. When these processes are combined, they engage in interaction as described in the scenario above.

2.4. Representing Communication (2): In-session Communication. An in-session communication specifies an operator and, as needed, a message content. First we present interaction without communication of values.

$$\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \mathtt{QuoteRequest} \rangle \,.\, G' \tag{4}$$

where *B2Sch* is an in-session channel. It says:

> Buyer selects *QuoteRequest*-branch of Seller, then the interaction G' ensues.

[1] For a concise theory, we do not have to declare this initialisation as types. However since it helps programmers in practice, we explicitly write them in the examples in this manuscript.

The same behaviour can be written down in the local calculus as:

$$\overline{B2Schs} \lhd \mathsf{QuoteRequest}\langle\rangle.P_1, \qquad B2Sch \rhd \mathsf{QuoteRequest}().P_2 \tag{5}$$

We use the slightly different notations from the main paper for readability:

- The construct $s \rhd \Sigma_i \mathsf{l}_i(x_i).P$ denotes an input action on the session channel s with options l_i. The received value will be stored in one of the variables x_i according to which branch is selected.
- Dually, the construct $\overline{s} \lhd \mathsf{l}\langle e\rangle.P$ denotes the action of outputting the value e on the session channel s with label l.
- **rec** $X.P$ denotes recursion.

Similarly, in global types, we write the label and sort types in one action; and denote $\Sigma_i G_i$ for branching.

An in-session communication may involve value passing, as follows.

$$\mathsf{Seller} \to \mathsf{Buyer} : \mathit{S2Bch}\,\langle \texttt{QuoteResponse}, \texttt{int}\rangle.G' \tag{6}$$

which says:

> *Seller selects a* QuoteRespons*-message with value with type* `int` *to Buyer; Buyer, upon reception, assigns the received value,* 3000*, to its some local variable x.*

(where the first argument QuoteRespons denotes the label while `int` denotes the type of the argument).

This description can be translated into end-point behaviours as follows.

$$\overline{S2Bch} \lhd \mathsf{QuoteResponse}\langle 3000\rangle.P_1, \qquad S2Bch \rhd \mathsf{QuoteResponse}(x).P_2 \tag{7}$$

which describes precisely the same communication behaviour.

2.5. Representing Branching. In various high-level protocols, we often find the situation where a sender invokes one of the options offered by a receiver. A method invocation in object-oriented languages is a simplest such example. In a global type, we may write an in-session communication which involves such a branching behaviour as follows.

$$\begin{array}{c} \{\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteAccept}\rangle.G_1\} \\ + \\ \{\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteReject}\rangle.G_2\} \end{array} \tag{8}$$

which reads:

> Through an in-session channel `B2Sch`, Buyer selects one of the two options offered by Seller, QuoteAccept and QuoteReject, and respectively proceeds to G_1 and G_2.

The same interaction can be written down in the local calculus as follows. First, Buyer's side (the one who selects) becomes:

$$\{\overline{B2Sch} \lhd \mathsf{QuoteAccept}\langle\rangle.P_1 \quad \text{or} \quad \overline{B2Sch} \lhd \mathsf{QuoteReject}\langle\rangle.P_2\} \tag{9}$$

In turn, Seller's side (which waits with two options) becomes:

$$\begin{array}{c} B2Sch \rhd \mathsf{QuoteAccept}().Q_1 \\ + \\ B2Sch \rhd \mathsf{QuoteReject}().Q_2 \end{array} \tag{10}$$

Here $+$ indicates this agent may either behave as $\mathit{B2Sch}\,\langle \texttt{QuoteAccept}\rangle.Q_1$ or as $\mathit{B2Sch}\,\langle \texttt{QuoteReject}\rangle.Q_2$ depending on what the interacting party communicates through B2Sch (this is so-called *branching*, whose nondeterminism comes from the behaviour of an external process). Note both branches start from input through the same channel B2Sch.

In the local descriptions, the original sum in the global type in (8) is decomposed into the selection and the branching. Similarly, G_1 (resp. G_2) may be considered as the result of interactions between P_1 and Q_1 (resp. P_2 and Q_2).

$$
\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Seller}, \mathsf{Shipper} : \mathsf{InitB2S}(\mathit{B2Sch}, \mathit{S2Hch})\,. \\
\mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteRequest} \rangle\,. \\
\mathsf{Seller} \to \mathsf{Buyer} : \mathit{B2Sch}\,\langle \texttt{QuoteResponse}, \texttt{quote} \rangle\,. \\
\{\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteAccept} \rangle\,. \\
\quad \mathsf{Seller} \to \mathsf{Buyer} : \mathit{B2Sch}\,\langle \texttt{OrderConfirmation} \rangle\,. \\
\quad \mathsf{Seller} \to \mathsf{Shipper} : \mathit{S2Hch}\,\langle \texttt{RequestDeliveryDetails} \rangle\,. \\
\quad \mathsf{Shipper} \to \mathsf{Seller} : \mathit{S2Hch}\,\langle \texttt{DeliveryDetails}, \texttt{details} \rangle\,. \\
\quad \mathsf{Seller} \to \mathsf{Buyer} : \mathit{B2Sch}\,\langle \texttt{DeliverDetails}, \texttt{details} \rangle\,.\,\texttt{end}\ \} \\
\qquad\qquad + \\
\{\ \mathsf{Buyer} \to \mathsf{Seller} : \mathit{B2Sch}\,\langle \texttt{QuoteReject} \rangle\,.\,\texttt{end}\ \}
\end{array}
$$

FIGURE 2. Global Description of Simple Protocol

2.6. Global Type of BSH Protocol with Multiparty Sessions. We can now present the whole of a global type of BSH Protocol, in Figure 2. While its meaning should be clear from our foregoing illustration, we illustrate the key aspects of the description in the following.

- Buyer initiates a session by invoking Seller through the session-initiating channel INITB2S, declaring two in-session channels `B2SCh` and `S2HCh`. Next, Buyer sends another message to Seller with the operation name "`QuoteRequest`" (which corresponds to the label name in the main paper) and without carried values (this message may as well be combined with the first one in practice).
- Seller then sends (and Buyer receives) a reply "`QuoteResponse`" together with the quote value with type `quote`. This received value will then be stored in x with type `quote`, local to Buyer.
- In the next step, Buyer decides whether the quote is acceptable or not. Accordingly:
  (1) Buyer may send `QuoteAccept`-message to Seller. Then Seller confirms the purchase, and asks Shipper for details of a delivery; Shipper answers with the requested details (say a delivery date), which Buyer forwards to Seller. Upon reception of this message the protocol terminates (denoted by `end`, the inaction).
  (2) Alternatively Buyer may send `QuoteReject`-message to Seller, in which case the protocol terminates without any further interactions.

Remark. If we do not use the multiparty sessions (i.e. binary sessions), we need to use another shared name for interaction between Seller and Shipper (i.e. for session *S2Sch*) so that we cannot have one global type, but two separated global types. This clearly shows that the multiparty sessions offer more expressive specifications for "more structured" communications than the binary sessions.

2.7. Local Description of Simple BSH Protocol. Figure 2 describes BSH Protocol from a vantage viewpoint, having all participants and their interaction flows in one view. The same behaviour can be described focussing on behaviours of individual participants, as follows (since we use explicit participants, we do not have to write initial participant as in the main paper).

The description is now divided into (1) Buyer's interactive behaviour, (2) Seller's interactive behaviour, and (3) Shipper's interactive behaviour. We focus on Buyer's behaviour. One can intuitively see two descriptions of the same protocol, a global version in Figure 2 and a local version in Figure 3, represent the same software behaviours — we can extract the former from the latter and vice versa.

$$
\begin{array}{ll}
\mathsf{Buyer}[& \overline{\mathsf{InitB2S}}\,[2,3](B2Sch,S2Hch)\,. \\
& \quad \overline{B2Sch} \lhd \mathsf{QuoteRequest}\langle\rangle\,. \\
& \quad B2Sch \rhd \mathsf{QuoteResponse}(x_{\mathtt{quote}})\,. \\
& \quad \{\ \overline{B2Sch} \lhd \mathsf{QuoteAccept}\langle.\rangle \\
& \qquad B2Sch \rhd \mathsf{OrderConfirmation}()\,. \\
& \qquad B2Sch \rhd \mathsf{DeliveryDetails}(y_{\mathtt{details}})\,.\,\mathbf{0}\ \} \quad] \\
\mathsf{Seller}[& \mathsf{InitB2S}\,[2](B2Sch,S2Hch)\,. \\
& \quad B2Sch \rhd \mathsf{QuoteRequest}()\,. \\
& \quad \overline{B2Sch} \lhd \mathsf{QuoteResponse}\langle v_{\mathtt{quote}}\rangle\,. \\
& \quad \{\ B2Sch \rhd \mathsf{QuoteAccept}()\,. \\
& \qquad \overline{B2Sch} \lhd \mathsf{OrderConfirmation}\langle\rangle\,. \\
& \qquad\quad \overline{S2Hch} \lhd \mathsf{DeliveryDetails}\langle\rangle\,. \\
& \qquad\quad S2Hch \rhd \mathsf{DeliveryDetails}(x_{\mathtt{details}})\,. \\
& \qquad \overline{B2Sch} \lhd \mathsf{DeliveryDetails}\langle x_{\mathtt{details}}\rangle\,.\,\mathbf{0}\ \} \\
& \qquad\quad + \\
& \quad \{\ B2Sch \rhd \mathsf{QuoteReject}()\,.\,\mathbf{0}\ \} \quad] \\
\mathsf{Shipper}[& \mathsf{InitB2H}\,[3](B2Sch,S2Hch)\,. \\
& \quad S2Hch \rhd \mathsf{DeliveryDetails}()\,. \\
& \quad \overline{S2Hch} \lhd \mathsf{DeliveryDetails}\langle v_{\mathtt{details}}\rangle\,.\,\mathbf{0} \quad]
\end{array}
$$

FIGURE 3. Local Description of Simple Protocol

A global type allows us to see how messages are exchanged between participants and how, as a whole, the interaction scenario proceeds; whereas, in the local description, the behaviour of each party is made explicit, as seen in distinct forms of choices used in Buyer and Seller.

3. Example 2

We now give a larger business protocol for multiparty session types. The example is an extension of the Buyer-Seller protocol. First, we give an informal description of the protocol.

3.1. Informal Description. There are five participants involved in this protocol:

Buyer (B), Seller (S), Vendor (V), CreditChecker (CC) and RoyalMail (R).

The purpose of the protocol is for Buyer to ask for a quote of a product to Seller, negotiates the price, and buys the product if its price is cheap enough and its credit card is valid. The negotiation process is done as a loop, which involves not only Buyer and Seller but also Vendor (which only interacts with Seller). When the negotiation is successful, Seller asks CreditChecker if Seller is credible, and if the answer is positive, asks RoyalMail (a shipper) to ship the good.

Remark. We stress that multiparty session types allow to reason about whom is required to take part to a session: if the type requires 3 participants then they all must synchronise (be present) in the initial hand-shake for the session to start. In this example we require that Buyer, Seller and the CrediChecker are participating to the protocol. Buyer is not aware of RoyalMail and Vendor which will be part of other sessions: it is up to Seller whether to contact somebody else for wholesale (Vendor) and postage (RoyalMail) or not. Using multiparty sessions, we can specify synchronisations between multi peers as types.

In total, the whole system involves three protocols (or sessions): one among Buyer, Seller and CreditChecker, one between Seller and RoyalMail and one between Seller and Vendor.

The protocol proceeds as follows:

(1) Buyer asks Seller for a quote about product prod;
(2) Seller then asks Vendor by opening a binary session ch_V
(3) Seller starts recursion and asks Vendor for a quote about product prod;
(4) Vendor replies with a quote quote;
(5) Seller forwards quote to Buyer increasing it by 10 units (quote+10);
(6) if the quote is reasonable ($reasonable(\mathsf{quote}+10)$) then:
- Buyer sends Seller a confirmation (QuoteOK) (which Seller forwards to Vendor) and sends its credit card details to CreditChecker;
- If the credit is good then:
  - CreditChecker contacts Seller
  - CreditChecker gives confirmation to Buyer;
  - Seller contacts RoyalMail (opening the binary session ch_{Sh});
  - Seller sends the delivery address;
  - RoyalMail sends a confirmation;
- If the credit is bad:
  - CreditChecker tells Seller and then Buyer and the protocol terminates;

(7) if the quote is not reasonable Buyer rejects the quote notifying Seller which notifies Vendor and protocol goes back to (3);

This concludes the informal presentation of the protocol global types.

3.2. Global Types. We now proceed as a programmer should: we give the global types for our three protocols.

- $ch_{BSCC}(bs, bcc, scc)$.

1. $B \to S : bs\langle \mathsf{Product} \rangle$.
2. **rec t** . {
3. $S \to B : bs\langle \mathsf{Quote} \rangle$.
4. $B \to S : bs\langle \mathsf{QuoteOK} \rangle$.
5. $B \to CC : bcc\langle \mathsf{CreditCard} \rangle$.
6. { $CC \to S : scc\langle \mathsf{Good} \rangle$.
7. { $CC \to B : bcc\langle \mathsf{Conf} \rangle$. end
8. +
9. $CC \to S : scc\langle \mathsf{Bad} \rangle$.
10. $CC \to B : bcc\langle \mathsf{YourCreditIsBad} \rangle$. end }
11. +
12. $B \to S : t\langle \mathsf{QuoteNotOK} \rangle$. **t** }

- $ch_{SV}(sv)$.

1. **rec t** . {
2. $S \to V : sv\langle \mathsf{QuoteReq} \rangle$.
3. $V \to S : sv\langle \mathsf{Quote} \rangle$.
4. $S \to V : sv\langle \mathsf{QuoteOK} \rangle$. end
5. +
6. $S \to V : sv\langle \mathsf{QuoteNotOK} \rangle$. t}

- $ch_{SR}(sr)$.

1. $S \to R : sr\langle \mathsf{Deliv} \rangle . R \to S : sr\langle \mathsf{Conf} \rangle$.

This concludes the formal representation of the protocol.

3.3. An running implementation in the end-point calculus. We now give the running code for the various participants involved satisfying the global types given above.
For readability reasons, we will give different processes for each participants which will then be composed in parallel. In the sequel, unimportant parameters in inputs and outputs will be omitted,

e.g. $s \triangleright \mathsf{op}$ stands for $s \triangleright \mathsf{op}(x)$ for some x. We start with Buyer:

$$
\begin{array}{l}
ch_{BSCC}[2](bs, bcc, scc)\,. \\
\overline{bs} \triangleleft \mathsf{Product}\langle prod \rangle\,. \\
\mathbf{rec}\, X\,. \\
bs \triangleright \mathsf{Quote}(quote)\,. \\
\mathsf{if}\ reasonable(quote)\ \mathsf{then} \\
\quad \bar{t} \triangleleft \mathsf{QuoteOK}\langle\rangle\,. \\
\quad \bar{t} \triangleleft \mathsf{CreditCard}\langle cred \rangle\,. \\
\quad t \triangleright \{\mathsf{YourCreditIsBad}\,.\,\mathbf{0} \ +\ \mathsf{Conf}\,.\,\mathbf{0}\} \\
\mathsf{else}\ \bar{t} \triangleleft \mathsf{QuoteNotOK}\,.\,X
\end{array}
$$

Note this process starts before the recursion and go through inside the (global) recursion. Thus this local behaviour also contains recursion. The next is a process of CreditChecker.

$$
\begin{array}{l}
ch_{BSCC}[3](bs, bcc, scc)\,. \\
s \triangleright \mathsf{CreditCard}(cred)\,. \\
\mathsf{if}\ validcard(cred)\ \mathsf{then} \\
\quad \overline{scc} \triangleleft \mathsf{Good}\langle\rangle\,. \\
\quad \overline{bcc} \triangleleft \mathsf{Conf}\langle details \rangle\,.\mathbf{0} \\
\mathsf{else} \\
\quad \overline{scc} \triangleleft \mathsf{Bad}\langle\rangle\,. \\
\quad \overline{bcc} \triangleleft \mathsf{YourCreditIsBad}\langle details \rangle\,.\mathbf{0}
\end{array}
$$

Note that both processes above do not include the recursion. This is because, in the global type, this part belongs to the not recursive branch (it is called only once the quote has been accepted). We now

move to the end-point implementation of Seller:

$$
\begin{array}{l}
\overline{ch_{BSCC}[2,3]}(\mathbf{v}\,bs, bcc, scc)\,. \\
t \triangleright \mathsf{QuoteReq}(prod)\,. \\
ch_{SV}[2](sv)\,. \\
\mathbf{rec}\,X\,. \\
\overline{sv} \triangleleft \mathsf{QuoteReq}\langle prod\rangle\,. \\
sv \triangleright \mathsf{Quote}(quote)\,. \\
\overline{bs} \triangleleft \mathsf{Quote}\langle quote + 10\rangle\,. \\
bs \triangleright \\
\quad \{\mathsf{QuoteOK}\,. \\
\quad\ bs \triangleright \mathsf{CreditCard}(cred)\,. \\
\quad\ \overline{sv} \triangleleft \mathsf{CreditCard}\langle cred\rangle\,. \\
\quad\ sv \triangleright \\
\quad\quad \{\mathsf{Good}\,. \\
\quad\quad\ ch_{SR}(sr)\,. \\
\quad\quad\ \overline{sr} \triangleleft \mathsf{Deliv}\langle prod\rangle\,. \\
\quad\quad\ sr \triangleright \mathsf{Conf}()\,.\,\mathbf{0} \\
\quad\quad\ + \\
\quad\quad\ \mathsf{Bad}\,.\,\mathbf{0}\} \\
\quad + \\
\quad\ \mathsf{QuoteNotOK}\,.\,X\}
\end{array}
$$

This process starts outside of the recursion in the global type and carries through the loop, so that both the recursion and the recursion variable are used as they are, leading to the recursive behaviour of the process. The process of Vendor follows.

$$
\begin{array}{l}
ch_{SV}[2](sv)\,. \\
\mathbf{rec}\,X\,. \\
sv \triangleright \mathsf{QuoteReq}(prod)\,. \\
\overline{sv} \triangleleft \mathsf{Quote}\langle quote\rangle\,. \\
sv \triangleright \\
\{\mathsf{QuoteOK}\,.\,\mathbf{0} \\
\quad + \\
\quad \mathsf{QuoteNotOK}\,.\,X\ \}
\end{array}
$$

Finally, we give the behaviour of RoyalMail.

$$ch_{SR}[2](sr)\,.\,sr \triangleright \mathsf{Deliv}(adr)\,.\,\overline{sr} \triangleleft \mathsf{Conf}$$

Bibliography

[1] G. Brown. A post at pi4soa forum. October, 2005.

[2] M. Carbone, K. Honda, and N. Yoshida. Theoretical basis of communication-centred concurrent programming (part two). http://www.dcs.qmul.ac.uk/∼carbonem/cdlpaper, June 2006.

[3] N. Kavantzas. A post at petri-pi mailing list. August, 2005.

[4] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, Sept. 1992.

[5] S. Ross-Talbot and T. Fletcher. Ws-cdl primer. Unpublished draft, May 2006.

[6] W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.