# Graph Representation For Global Protocol In Generalised Multiparty Session Types

Background paper

Charlotte Pichot,
MSc in Advanced Computing
Supervisor: Dr. Nobuko Yoshida
Department of Computing, Imperial College London

June 8, 2012

**Abstract**

This paper reports all the primarily work I have done for my individual project. The goal of my project is to develop a tool to get a graph representation of global protocol in generalised multiparty session types. For the moment, a lot of research work is done on multiparty session types. Its purpose is to provide a safe structure, with global and local types, to model interactions between processes. However, there is no tool in the literature to graphically represent global protocols. Therefore we will discuss this aspect. This report contains the work I have done so far with a literature review on the subject and a plan for the future work.

## Introduction

We have seen such a development in the area of distributed networks for the past few years, for instance the web services. Networks are continuously growing with all the connected devices that appear in our daily life: there are not only computers and mobile phones, but also smart devices, like smart meter for the electric grid.
To be connected altogether, they need to have communication features. That is why it brings new challenges for developers. As soon as communication is concerned, we must ensure some fundamental properties as safety, that is there should be no deadlocks, no mismatchs and processes should be able to progress.
In the course Models of Concurrent Computation, I discovered some ways of modelling interactions between processes with the study of CCS and Pi-calculus. To go further in this very interesting topic, I chose to do my individual project with Dr Nobuko Yoshida, one of the lecturer of this course. Therefore it enables me to take part in the research in this field.
The current research states a new structure, called a session, and an associated typing system to safely model communication in distributed networks. Besides this structure, developers need a real tool chain: from the graph representation of the global procotol to the code. I focus on the graph representation as automata for my project.
To understand the purpose of generalised multiparty session types, we first need to get knowledge on binary session types and then multiparty session types. From this basis, I introduce my project, set the first steps I went through and detail the contribution I plan to achieve.

## 1 Motivation example

We consider a real world use case ([1]) to introduce the purpose of the project and the notion of graph representation of a global protocol. The use case is as follow: UC.R2.13 “Acquire Data From Instrument” from the Ocean Observatories Initiative (OOI) Use Case library (Release 2).
This use case describes a scenario where a user program (U) is connected to the Integrated Observatory Network (ION), which provides the infrastructure between users and remote sensing instruments.
The user requests, via an ION agent service (A), the acquisition of data from an instrument (I), e.g. a temperature sensor, registered with the ION. One of two data acquisition methods can be requested by U. One is push mode, where the data is streamed by I (driven by I’s clock). The other is poll mode, where the data is periodically polled by A (driven by A’s clock) on behalf of U. However, I may support only one of these modes. If not supported, A is responsible for emulating the requested mode over the available one. In both modes, A formats and relays the acquired data to U as a stream; U can interrupt the stream at any point to end the conversation.

First to get the idea of the protocol, we draw a diagram of the choreography (Figure 1).

As we are interested in global protocol, we present in Figure 2 the translation of this use case into global types as defined in generalised mutliparty session types. We can

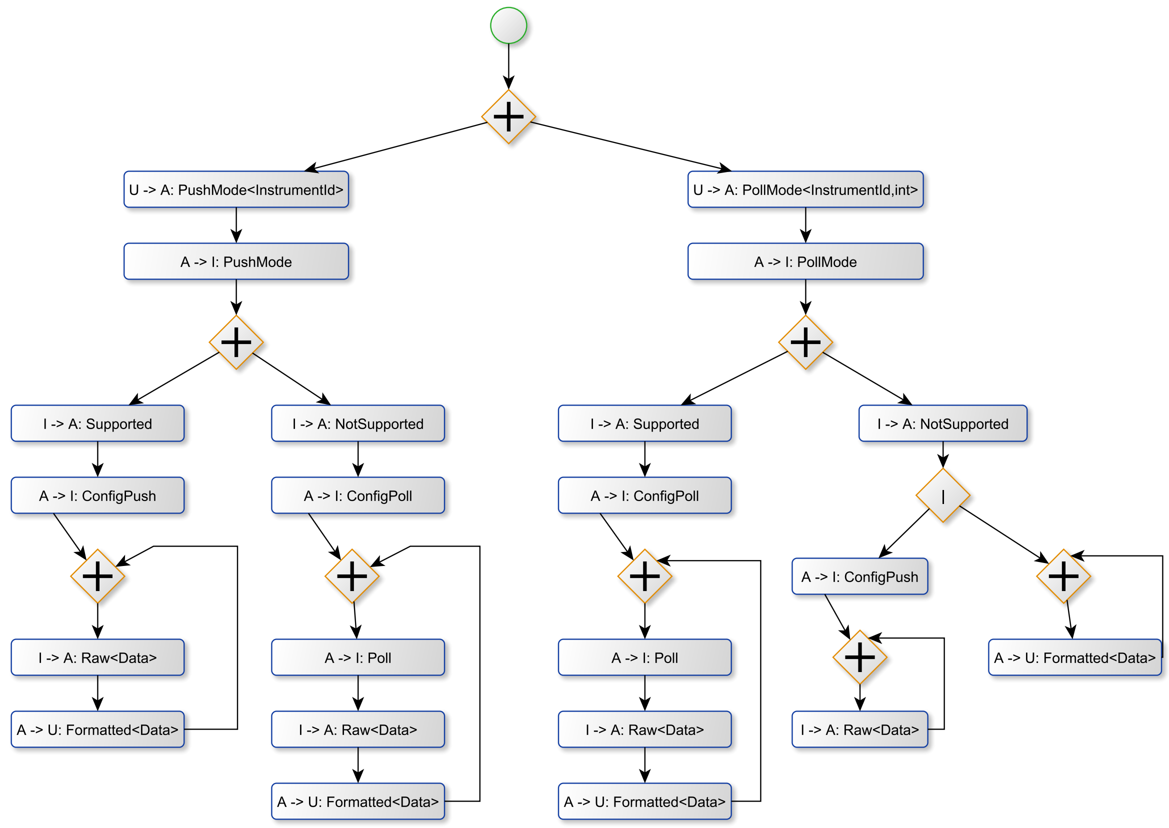

Figure 1: The choreography for the OOI "Acquire Data from Instrument" use case

easily see the exact match between the graph representation and the global protocol: indeed, edges are variables and nodes are transitions.
$x_0$ represents the initial state.
$x_0 = x_{push} + x_{poll}$ is a choice, that is U will make the choice between continuing with $x_{push}$ or $x_{poll}$.
$x_{pushsup2} + x_{pushsup4} = \text{I} \rightarrow \text{A: Raw}\langle\text{Data}\rangle; x_{pushsup3}$ is a merge and in this case expresses recursion.
$x_{pollnsup1} = x_{pollnsup2} \mid x_{pollnsup3}$ is a fork to model interleaving of actions at $x_{pollnsup2}$ and $x_{pollnsup3}$.

Once global types have been established, we can write the corresponding safe program, using Scribble as language. The program for this example is given in Figure 3. Then there is a tool to deduce from the Scribble program the safe local programs.

Therefore the missing part of the tool chain is the graph representation of the globol protocol, the development of which will be our contribution. From now on, we will

$$
\begin{array}{llrcl}
\text{G} = & \text{def} & x_0 & = & x_{push} + x_{poll} \\
 & & x_{push} & = & \text{U} \rightarrow \text{A: PushMode}\langle \text{ InstrumentId } \rangle; x_{push1} \\
 & & x_{push1} & = & \text{A} \rightarrow \text{I: PushMode }; x_{push2} \\
 & & x_{push2} & = & x_{pushsup} + x_{pushnsup} \\
 & & x_{pushsup} & = & \text{I} \rightarrow \text{A: Supported}; x_{pushsup1} \\
 & & x_{pushsup1} & = & \text{A} \rightarrow \text{I: ConfigPush}; x_{pushsup2} \\
 & & x_{pushsup2} + x_{pushsup4} & = & \text{I} \rightarrow \text{A: Raw}\langle \text{ Data } \rangle; x_{pushsup3} \\
 & & x_{pushsup3} & = & \text{A} \rightarrow \text{U: Formatted}\langle \text{ Data } \rangle; x_{pushsup4} \\
 & & x_{pushnsup} & = & \text{I} \rightarrow \text{A: NotSupported}; x_{pushnsup1} \\
 & & x_{pushnsup1} & = & \text{A} \rightarrow \text{I: ConfigPoll}; x_{pushnsup2} \\
 & & x_{pushnsup2} + x_{pushnsup5} & = & \text{A} \rightarrow \text{I: Poll}; x_{pushnsup3} \\
 & & x_{pushnsup3} & = & \rightarrow \text{A: Raw}\langle \text{ Data } \rangle; x_{pushnsup4} \\
 & & x_{pushnsup4} & = & \text{A} \rightarrow \text{U: Formatted}\langle \text{ Data } \rangle; x_{pushnsup5} \\
 & & x_{poll} & = & \text{U} \rightarrow \text{A: PollMode}\langle \text{ InstrumentId, int } \rangle; x_{poll1} \\
 & & x_{poll1} & = & \text{A} \rightarrow \text{I: PollMode }; x_{poll2} \\
 & & x_{poll2} & = & x_{pollsup} + x_{pollnsup} \\
 & & x_{pollsup} & = & \text{I} \rightarrow \text{A: Supported}; x_{pollsup1} \\
 & & x_{pollsup1} & = & \text{A} \rightarrow \text{I: ConfigPoll}; x_{pollsup2} \\
 & & x_{pollsup2} + x_{pollsup5} & = & \text{A} \rightarrow \text{I: Poll}; x_{pollsup3} \\
 & & x_{pollsup3} & = & \rightarrow \text{A: Raw}\langle \text{ Data } \rangle; x_{pollsup4} \\
 & & x_{pollsup4} & = & \text{A} \rightarrow \text{U: Formatted}\langle \text{ Data } \rangle; x_{pollsup5} \\
 & & x_{pollnsup} & = & \text{I} \rightarrow \text{A: NotSupported}; x_{pollnsup1} \\
 & & x_{pollnsup1} & = & x_{pollnsup2} \mid x_{pollnsup3} \\
 & & x_{pollnsup2} & = & \text{A} \rightarrow \text{I: ConfigPush}; x_{pollnsup4} \\
 & & x_{pollnsup4} + x_{pollnsup5} & = & \text{I} \rightarrow \text{A: Raw}\langle \text{ Data } \rangle; x_{pollnsup5} \\
 & & x_{pollnsup3} + x_{pollnsup6} & = & \text{A} \rightarrow \text{U: Formatted}\langle \text{ Data } \rangle; x_{pollnsup6} \\
 & \text{in } x_0 & & &
\end{array}
$$

Figure 2: The global protocol for the OOI "Acquire Data from Instrument" use case

review the different calculus to explain the origin of generalised multiparty session types.

## 2 Session Types

As we have seen in the introduction there is a need in distributed programming for a clear structure to express conversation.
A session is a structure to encapsulate a safe communication scheme between two or more peers in a context of distributed processes. A type system is associated to this structure in order to statically check the programs from the processes. A session is introduced to describe a sequence of interactions between the processes.

As the case of two processes was first studied, we begin with exposing the binary session types to get the idea of session types. Then it has been extended to more than two processes with multiparty session types in order to express more complex interactions. Therefore we will continue with a detailed presentation of multiparty

```
// U is User, A is ION Agent, I is Instrument
global protocol DataAcquisition(role U, role A, role I) {
  try {  choice at U {
            PushMode(InstrumentId) from U to A;
            PushMode from A to I;
            choice at I {
              Supported from I to A;
              ConfigPush from A to I;
              rec PUSH {
                Raw(Data) from I to A;
                Formatted(Data) from A to U;
                PUSH;
              }
            } or {
              NotSupported from I to A;
              ConfigPoll from A to I;
              rec POLL {
                Poll from A to I;
                Raw(Data) from I to A;
                Formatted(Data) from A to U;
                POLL;
            } }
          } or {
            PollMode(InstrumentId, int) from U to A;
            PollMode from A to I;
            choice at I {
              Supported from I to A;
              ConfigPoll from A to I;
              rec POLL {
                Poll from A to I;
                Raw(Data) from I to A;
                Formatted(Data) from A to U;
                continue POLL;
              }
            } or {
              NotSupported from I to A;
              parallel {
                ConfigPush from A to I;
                rec PUSH {
                  Raw(Data) from I to A;
                  continue PUSH;
                }
              } and {
                rec POLL {
                  Formatted(Data) from A to U;
                  continue POLL; }
            } }
          }
  } interrupt {
    Stop by U
} }
```

Figure 3: The Scribble global protocol for the OOI “Acquire Data from Instrument” use case

|  |  |  |  |
|---|---|---|---|
| $P$ | ::= | request a(k) in $P$ | Session Request |
|  | $\mid$ | accept a(k) in $P$ | Session Acceptance |
|  | $\mid$ | $k!\langle e\rangle;P$ | Data sending |
|  | $\mid$ | $k?(x);P$ | Data reception |
|  | $\mid$ | throw $k$[$k$']; $P$ | Channel Sending |
|  | $\mid$ | catch $k$($k$') in $P$ | Channel Reception |
|  | $\mid$ | $k\lhd$ ;$P$ | Label Selection |
|  | $\mid$ | $k\rhd\ \{l_1 : P_1\|...\|l_n : P_n\}$ | Label Branching |
|  | $\mid$ | if $e$ then $P$ else $Q$ | Conditional Branch |
|  | $\mid$ | $P \mid Q$ | Parallel |
|  | $\mid$ | inact | Inaction |
|  | $\mid$ | $(\nu u)P$ | Name/Channel Hiding |
|  | $\mid$ | def $D$ in $P$ | Recursion |
|  | $\mid$ | $X[ek]$ | Process Variables |
| $e$ | ::= | $c$ | Constant |
|  | $\mid$ | $e$+$e$' $\mid$ $e$-$e$' $\mid$ $e$+$e$' $\mid$ not $e$ ...x | Expression |
| $D$ | ::= | $X(x,y) = P$ | Declaration |

Table 1: Syntax for user-defined processes

session.

## 2.1 Binary Session Types

The binary session types contains the expected communication features, as studied in the Models of Concurrent Computation course: value sending and receiving, recursion, choice of interactions. What is new is the session inititation and session delegation.

We take an example to explain in more details the syntax (Table 1), operational semantics and typing system. We consider the case of a client using the National Rail website to book a train ticket. The scenario is as follow:

- The client first choose the destination.
- Then the service contact the right company to delegate the deal. The client will continue the transaction, unaware that he now communicates directly with the company.
- The client and the company exchange date, price and money.

Here is the syntax for the system made up of one client, *Client*, the National Rail Services, *NRS*, and one of the companies, *Company*.

def Client = request a(k) in k $\lhd$ bristol;k!(date);k?(price);k!(money);inact
and NRS(u,v) = accept u(v) in k $\rhd$ {bristol:request c(k') in throw k'[v];NRS(u',v') ||...|| newcastle: ... }

and Company = accept c(k') in catch k'[k] in k?(date);k!(price);k?(money);Company
in Client | NRS(a,k) | Company

The first line specifies that a client first requests a session withe the national rail services, then chooses a destination, here labelled Bristol, and finally sends the date, receives the corresponding price and sends the money before halting. The second line describes the interactions from the national rail side: it accepts the request of session and then have a branching choice. We only specify the case for the label called Bristol: NRS requests a new session with the corresponding company and delegates the deal, before being available again for other interactions. The third line specifies that a company first accepts the request for a new session, then receives the delegation of the deal and finally operates the receiveing of the date, the sending of the price and the receiving of the money, before being available again.

Rather than exposing all the tables for operational semantics and typing system, that can be found in [5, 8], we apply the rules to reduct our example.
Let call $P$ the system above defined and D the declaration for recursion inside $P$, and apply to it the reduction rules.

$P$ $\rightarrow$ def D in (ν k) ( k $\triangleleft$ bristol;k!(date);k?(price);k!(money);inact | [Link]
k $\triangleright$ {bristol:request c(k') in throw k'[k];NRS(u',v') $\|$...$\|$ newcastle: ... }) |
accept c(k') in catch k'[k] in k?(date);k!(price);k?(money);Company

$\rightarrow$ def D in (ν k) (k!(date);k?(price);k!(money);inact | [Label]
request c(k') in throw k'[k];NRS(u',v') )|
accept c(k') in catch k'[k] in k?(date);k!(price);k?(money);Company

$\rightarrow$ def D in (ν k) (k!(date);k?(price);k!(money);inact | [Link]
(ν k')(throw k'[k];NRS(u',v') |
catch k'[k] in k?(date);k!(price);k?(money);Company))

$\rightarrow$ def D in (ν k) (k!(date);k?(price);k!(money);inact | [Pass]
(ν k')(NRS(u',v') | k?(date);k!(price);k?(money);Company))

$\rightarrow$ def D in (ν k) (k?(price);k!(money);inact | (ν k')(NRS(u',v') |
k!(price);k?(money);Company)) [Com]

$\rightarrow$ def D in (ν k)( k!(money);inact | (ν k')(NRS(u',v') | k?(money);Company)) [Com]

$\rightarrow$ def D in (ν k) (inact | (ν k')(NRS(u',v') | Company)) [Com]

$\equiv$ def D in (NRS(u',v') | Company)

Besides the specified reduction rules we used [Def] at each step and [Str] at the second [Link] step in order to put the third part inside (ν k).

The process $P$ is also typable with respect to the typing system. We give here two steps of this typing:

$$\Gamma \vdash k \triangleleft bristol; k!(date); k?(price); k!(money); inact \triangleright \Delta, k : \oplus\{bristol : ![nat]; ?[nat]; ![nat]; end\}$$

$$\Gamma \vdash catch\, k'[k] in\, k?(date); k!(price); k?(money); inact \triangleright \Delta, k' : ?[\ ![nat]; ?[nat]; ![nat]; end\ ]; end$$

Therefore binary session types allows to express complex communication structure between two peers and to ensure its safety. Nevertheless, in a real world context, communication usually involves more than two entities. To solve this issue, one can first think of modelling interactions between more than two processes using binary session types for each pair of them. However it appears to remove the desired clarity of the structure. Furthermore it can not express the situation as a whole but only separate interactions. That is why researchers introduce multiparty session types.

## 2.2 Multiparty Session Types

### 2.2.1 Overview

A multiparty session describe the interactions between several processes. Besides specifying the user-defined processes involved in a session, we have to define, at a higher level, a global protocol that ensures the coherence of the session. The type system allows specifying these two levels: first there is a global type and then projections of this global type to each endpoint processes.
In multiparty session programming, a developer will have to define the global type and the user-defined processes. The key step in the methodology consists then of checking the correspondence between the projections from the global type and the type of the endpoint programs.

With respect to the procedure, we get several properties: type safety, session fidelity, progress, linearity. Also the structure of sessions makes it possible to deal with interferences and interleaving of interactions among participants.

To describe with more details and more formally what is explained above, we look at the example of the three buyers protocol and then introduce syntax, operational semantics and a type system, as defined in [2].

### 2.2.2 The three buyers protocol example

We first introduce the calculus through the example of the three-buyer protocol, which includes the expected communiaction features, as well as session-multicasting and dynamically merging two conversations. The overall scenario proceeds as follows.

1. Alice sends a book title to Seller, then Seller sends back a quote to Alice and Bob. Then Alice tells Bob how much she can contribute.

2. If the price is within Bob's budget, Bob notifies both Seller and Alice he accepts, then sends his address, and Seller sends back the delivery date.

3. If the price exceeds the budget, Bob asks Carol to collaborate together by establishing a new session. Then Bob sends how much Carol must pay, then *delegates* the remaining interactions with Alice and Seller to Carol.

4. If the rest of the price is within Carol's budget, Carol accepts the quote and notifies Alice, Bob and Seller, and continues the rest of the protocol with Seller and Alice transparently, *as if she were Bob*. Otherwise she notifies Alice, Bob and Seller to quit the protocol.

Figure 4 depicts an execution of the above protocol where Bob asks Carol to collaborate (by delegating the remaining interactions with Alice and Seller) and the transaction terminates successfully.

Then multiparty session programming consists of two steps: specifying the intended communication protocols using global types, and implementing these protocols

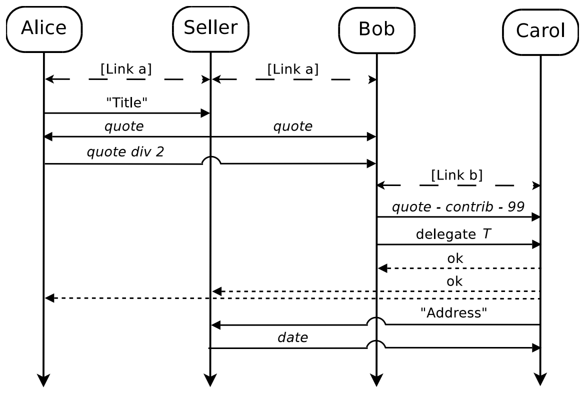

Figure 4: The three buyer protocol interactions

using processes. The specifications of the three-buyer protocol are given as two separated global types: one is $G_a$ among Alice, Bob and Seller and the other is $G_b$ between Bob and Carol. We write principals with legible symbols though they will actually be coded by numbers: in $G_a$ we have $\mathtt{A} = 1$, $\mathtt{B} = 2$, and $\mathtt{S} = 3$, while in $G_b$ we have $\mathtt{B} = 2$, $\mathtt{C} = 1$.

$G_a =$

```
1.  A  ⟶  S :      ⟨string⟩.
2.  S  ⟶  {A,B} :  ⟨int⟩.
3.  A  ⟶  B :      ⟨int⟩.
4.  B  ⟶  {S,A} :  {ok :B ⟶ S : ⟨string⟩.
5.                          S ⟶ B : ⟨date⟩;end
6.                  quit : end}
```

$G_b =$

```
1.  B  ⟶  C :  ⟨int⟩.
2.  B  ⟶  C :  ⟨T⟩.
3.  C  ⟶  B :  {ok : end,   quit : end}.
      T =
      ⊕⟨{S,A},
        {ok :!⟨S,string⟩; ?⟨S,date⟩; end,
         quit : end}⟩
```

The types give a global view of the two conversations, directly abstracting the scenario given by the diagram. In $G_a$, line 1 denotes A sends a string value to S. Line 2 says S multicasts the same integer value to A and B and line 3 says that A sends an integer to B. In lines 4-6 B sends either ok or quit to S and A. In the first case B sends a string to S and receives a date from S, in the second case there are no further communications.

Line 2 in $G_b$ represents the delegation of the capability specified by the session type $T$ of channels (formally defined later) from B to C (note that S and A in $T$ concern the session on $a$).

Figure 5 gives the code, associated to $G_a$ and $G_b$, for S, A, B and C in a "user" syntax formally defined later[1]:

Session name $a$ establishes the session corresponding to $G_a$. S initiates a session involving three bodies as third participant by $\overline{a}[3](y_3)$: A and B participate as first and second participants by $a[1](y_1)$ and $a[2](y_2)$, respectively. Then S, A and B commu-

[1]In the examples we will use the following font conventions: variables (bound by an input action) are in *italics* and constants are in sans serif; string literals are in `monospace` font and double quoted.

$$
\begin{array}{lcl}
\mathtt{S} & = & \overline{a}[3](y_3).y_3?(\mathit{title});y_3!\langle \mathsf{quote}\rangle;y_3\&\{\mathsf{ok}: y_3?(\mathit{address});y_3!\langle\mathsf{date}\rangle;\mathbf{0},\ \mathsf{quit}:\mathbf{0}\}\\
\mathtt{A} & = & a[1](y_1).y_1!\langle \texttt{"Title"}\rangle;y_1?(\mathit{quote});y_1!\langle \mathit{quote}\ \mathsf{div}\ 2\rangle;y_1\&\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\}\\
\mathtt{B} & = & a[2](y_2).y_2?(\mathit{quote});y_2?(\mathit{contrib});\\
 & & \mathsf{if}\ (\mathit{quote}\ \text{-}\ \mathit{contrib} < 100)\ \mathsf{then}\ \ y_2\oplus\mathsf{ok};y_2!\langle\texttt{"Address"}\rangle;y_2?(\mathit{date});\mathbf{0}\\
 & & \mathsf{else}\ \overline{b}[2](z_2).z_2!\langle \mathit{quote}\ \text{-}\ \mathit{contrib}\ \text{-}\ 99\rangle;z_2!\langle\langle y_2\rangle\rangle;z_2\&\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\}\\
\mathtt{C} & = & b[1](z_1).z_1?(x);z_1?((t));\\
 & & \mathsf{if}\ (x<100)\ \mathsf{then}\ z_1\oplus\mathsf{ok};t\oplus\mathsf{ok};t!\langle\texttt{"Address"}\rangle;t?(\mathit{date});\mathbf{0}\\
 & & \mathsf{else}\ z_1\oplus\mathsf{quit};t\oplus\mathsf{quit};\mathbf{0}
\end{array}
$$

Figure 5: The three buyer example

nicate using the channels $y_3$, $y_1$ and $y_2$, respectively. Each channel $y_\mathtt{p}$ can be seen as a port connecting participant p with all other ones; the receivers of the data sent on $y_\mathtt{p}$ are specified by the global type (this information will be included in the runtime code). The first line of $G_a$ is implemented by the input and output actions $y_3?(\mathit{title})$ and $y_1!\langle\texttt{"Title"}\rangle$. The last line of $G_b$ is implemented by the branching and selection actions $z_2\&\{\mathsf{ok}:\mathbf{0},\ \mathsf{quit}:\mathbf{0}\}$ and $z_1\oplus\mathsf{ok}$, $z_1\oplus\mathsf{quit}$.

In B, if the quote minus A's contribution exceeds 100€ (i.e., $\mathit{quote}\ \text{-}\ \mathit{contrib} \geq 100$), another session between B and C is established dynamically through shared name $b$. The delegation is performed by passing the channel $y_2$ from B to C (actions $z_2!\langle\langle y_2\rangle\rangle$ and $z_1?((t))$), and so the rest of the session is carried out by C with S and A.

This example illustrates how multiparty session types is used for such a protocol and garantees safe communication.

### 2.2.3 Syntax and Operational semantics

The real improvement in multiparty session types, compared to binary session types, is the notion of global protocol, which will be used as a choreography for the whole system. Ths syntax for global protocols can be found in [2]. The grammar for processes, ranged over by $P, Q\ldots$, and for expressions, ranged over by $e, e', \ldots$, similar to the binary session case, is given by the Table 2.
The rules for operational semantics are also from [2] and given in Tables **??** 4.

### 2.2.4 Typing System

As we have introduced the notion of global protocol, we now define global types. As explained in the three buyers protocol, from the global type, one can deduce the local types thank to projections.

A *global type*, ranged over by $G, G'$, .. describes the whole conversation scenario of a multiparty session as a type signature. Its grammar is given below:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $P$ | $::=$ | $\overline{u}[\mathrm{p}](y).P$ | Multicast Request | | $\mid$ | $\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$ | Conditional |
| | $\mid$ | $u[\mathrm{p}](y).P$ | Accept | | $\mid$ | $P \mid Q$ | Parallel |
| | $\mid$ | $y!\langle e\rangle;P$ | Value sending | | $\mid$ | $\mathbf{0}$ | Inaction |
| | $\mid$ | $y?(x);P$ | Value reception | | $\mid$ | $(\nu a)P$ | Hiding |
| | $\mid$ | $y!\langle\langle z\rangle\rangle;P$ | Session delegation | | $\mid$ | $\mathsf{def}\ D\ \mathsf{in}\ P$ | Recursion |
| | $\mid$ | $y?((z));P$ | Session reception | | $\mid$ | $X\langle e,y\rangle$ | Process call |
| | $\mid$ | $y \oplus l;P$ | Selection | | | | |
| | $\mid$ | $y\&\{l_i : P_i\}_{i\in I}$ | Branching | $e$ | $::=$ | $v \mid x$ | |
| $u$ | $::=$ | $x \mid a$ | Identifier | | $\mid$ | $e\ \mathsf{and}\ e' \mid \mathsf{not}\ e \ldots$ | Expression |
| $v$ | $::=$ | $a \mid \mathsf{true} \mid \mathsf{false}$ | Value | $D$ | $::=$ | $X(x,y) = P$ | Declaration |

Table 2: Syntax for user-defined processes

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$$

$$(\nu r)P \mid Q \equiv (\nu r)(P \mid Q) \qquad \text{if } r \notin \mathrm{fn}(Q)$$

$$(\nu r r')P \equiv (\nu r' r)P \qquad (\nu r)\mathbf{0} \equiv \mathbf{0} \qquad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0}$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\nu r)P \equiv (\nu r)\mathsf{def}\ D\ \mathsf{in}\ P \qquad \text{if } r \notin \mathrm{fn}(D)$$

$$(\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \equiv \mathsf{def}\ D\ \mathsf{in}\ (P \mid Q) \qquad \text{if } \mathrm{dpv}(D) \cap \mathrm{fpv}(Q) = \emptyset$$

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ P) \equiv \mathsf{def}\ D\ \mathsf{and}\ D'\ \mathsf{in}\ P \qquad \text{if } \mathrm{dpv}(D) \cap \mathrm{dpv}(D') = \emptyset$$

$$s : (\mathrm{q},\Pi,z)\cdot(\mathrm{q}',\Pi',z')\cdot h \equiv s : (\mathrm{q}',\Pi',z')\cdot(\mathrm{q},\Pi,z)\cdot h \qquad \text{if } \Pi \cap \Pi' = \emptyset \text{ or } \mathrm{q} \neq \mathrm{q}'$$

$$s : (\mathrm{q},\Pi,z)\cdot h \equiv s : (\mathrm{q},\Pi',z)\cdot(\mathrm{q},\Pi'',z)\cdot h \qquad \text{where } \Pi = \Pi' \cup \Pi'' \text{ and } \Pi' \cap \Pi'' = \emptyset$$

Table 3: Structural equivalence ($r$ ranges over $a$, $s$ and $z$ ranges over $v$, $s[\mathrm{p}]$ and $l$.)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Global | $G$ | $::=$ | $\mathrm{p} \to \Pi : \langle U\rangle.G'$ | Exchange | $U$ | $::=$ | $S \mid T$ |
| | | $\mid$ | $\mathrm{p} \to \Pi : \{l_i : G_i\}_{i\in I}$ | Sorts | $S$ | $::=$ | $\mathsf{bool} \mid \ldots \mid G$ |
| | | $\mid$ | $\mu\mathbf{t}.G \mid \mathbf{t} \mid \mathsf{end}$ | | | | |

We now define the local types of pure processes, called *session types*. While global types represent the whole protocol, session types correspond to the communication actions, representing sessions from the view-points of single participants.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Action | $T$ | $::=$ | $!\langle\Pi,U\rangle;T$ | *send* | $\mid$ | $\mu\mathbf{t}.T$ | *recursive* |
| | | $\mid$ | $?(\mathrm{p},U);T$ | *receive* | $\mid$ | $\mathbf{t}$ | *variable* |
| | | $\mid$ | $\oplus\langle\Pi,\{l_i : T_i\}_{i\in I}\rangle$ | *selection* | $\mid$ | $\mathsf{end}$ | *end* |
| | | $\mid$ | $\&(\mathrm{p},\{l_i : T_i\}_{i\in I})$ | *branching* | | | |

The relation between session and global types is formalised by the notion of projection. The *projection of* $G$ *onto* q ($G \restriction \mathrm{q}$) is defined by induction on $G$:

$$a[1](y_1).P_1 \mid ... \mid \overline{a}[n](y_n).P_n \longrightarrow (\nu s)(P_1\{s[1]/y_1\} \mid ... \mid P_n\{s[n]/y_n\} \mid s:\varnothing) \qquad \text{[Link]}$$

$$s[\mathrm{p}]!\langle \Pi, e\rangle;P \mid s:h \longrightarrow P \mid s:h\cdot(\mathrm{p},\Pi,v) \quad (e{\downarrow}v) \qquad \text{[Send]}$$

$$s[\mathrm{p}]!\langle\langle \mathrm{q}, s'[\mathrm{p}']\rangle\rangle;P \mid s:h \longrightarrow P \mid s:h\cdot(\mathrm{p},\mathrm{q},s'[\mathrm{p}']) \qquad \text{[Deleg]}$$

$$s[\mathrm{p}]\oplus\langle \Pi, l\rangle;P \mid s:h \longrightarrow P \mid s:h\cdot(\mathrm{p},\Pi,l) \qquad \text{[Label]}$$

$$s[\mathrm{p}]?(\mathrm{q},x);P \mid s:(\mathrm{q},\{\mathrm{p}\},v)\cdot h \longrightarrow P\{v/x\} \mid s:h \qquad \text{[Recv]}$$

$$s[\mathrm{p}]?((\mathrm{q},y));P \mid s:(\mathrm{q},\mathrm{p},s'[\mathrm{p}'])\cdot h \longrightarrow P\{s'[\mathrm{p}']/y\} \mid s:h \qquad \text{[Srec]}$$

$$s[\mathrm{p}]\&(\mathrm{q},\{l_i:P_i\}_{i\in I}) \mid s:(\mathrm{q},\{\mathrm{p}\},l_{i_0})\cdot h \longrightarrow P_{i_0} \mid s:h \quad (i_0\in I) \qquad \text{[Branch]}$$

$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow P \quad (e\downarrow \mathsf{true}) \quad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow Q \quad (e\downarrow \mathsf{false}) \qquad \text{[If-T, If-F]}$$

$$\mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (X\langle e, s[\mathrm{p}]\rangle \mid Q) \longrightarrow \mathsf{def}\ X(x,y)=P\ \mathsf{in}\ (P\{v/x\}\{s[\mathrm{p}]/y\} \mid Q) \quad (e\downarrow v) \qquad \text{[ProcCall]}$$

$$P\longrightarrow P' \quad\Rightarrow\quad (\nu r)P \longrightarrow (\nu r)P' \qquad P\longrightarrow P' \quad\Rightarrow\quad P\mid Q \longrightarrow P'\mid Q \qquad \text{[Scop,Par]}$$

$$P\longrightarrow P' \quad\Rightarrow\quad \mathsf{def}\ D\ \mathsf{in}\ P \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ P' \qquad \text{[Defin]}$$

$$P\equiv P' \text{ and } P'\longrightarrow Q' \text{ and } Q\equiv Q' \quad\Rightarrow\quad P\longrightarrow Q \qquad \text{[Str]}$$

Table 4: Reduction rules

$$(\mathrm{p}\to\Pi:\langle U\rangle.G')\restriction\mathrm{q} = \begin{cases} !\langle \Pi, U\rangle;(G'\restriction\mathrm{q}) & \text{if } \mathrm{q}=\mathrm{p},\\ ?(\mathrm{p},U);(G'\restriction\mathrm{q}) & \text{if } \mathrm{q}\in\Pi,\\ G'\restriction\mathrm{q} & \text{otherwise.}\end{cases}$$

$$(\mathrm{p}\to\Pi:\{l_i:G_i\}_{i\in I})\restriction\mathrm{q} = \begin{cases} \oplus(\Pi,\{l_i:G_i\restriction\mathrm{q}\}_{i\in I}) & \text{if } \mathrm{q}=\mathrm{p}\\ \&(\mathrm{p},\{l_i:G_i\restriction\mathrm{q}\}_{i\in I}) & \text{if } \mathrm{q}\in\Pi\\ G_1\restriction\mathrm{q} & \text{if } \mathrm{q}\neq\mathrm{p}, \mathrm{q}\notin\Pi \text{ and}\\ & G_i\restriction\mathrm{q}=G_j\restriction\mathrm{q} \text{ for all } i,j\in I.\end{cases}$$

$$(\mu\mathbf{t}.G)\restriction\mathrm{q} = \mu\mathbf{t}.(G\restriction\mathrm{q}) \quad \mathbf{t}\restriction\mathrm{q}=\mathbf{t} \quad \mathsf{end}\restriction\mathrm{q}=\mathsf{end}.$$

As an example, we list two of the projections of the global types $G_a$ and $G_b$ of the first three-buyer protocol:

$$\begin{aligned} G_a\restriction 1 &= !\langle\{3\},\mathsf{string}\rangle;?(3,\mathsf{int});!\langle\{2\},\mathsf{int}\rangle;\&(2,\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\})\\ G_b\restriction 2 &= !\langle\{1\},\mathsf{int}\rangle;!\langle\{1\},T\rangle;\&(1,\{\mathsf{ok}:\mathsf{end},\mathsf{quit}:\mathsf{end}\})\end{aligned}$$

where $T=\oplus\langle\{1,3\},\{\mathsf{ok}:!\langle\{3\},\mathsf{string}\rangle;?\langle 3,\mathsf{date}\rangle;\mathsf{end},\ \mathsf{quit}:\mathsf{end}\}\rangle$.

The typing judgements for expressions and pure processes are of the shape:

$$\Gamma\vdash e:S \quad\text{and}\quad \Gamma\vdash P\triangleright\Delta$$

where $\Gamma$ is the *standard environment* which associates variables to sort types, service names to global types and process variables to pairs of sort types and session types;

$$\Gamma, u : S \vdash u : S \ \lfloor\textsc{Name}\rfloor \qquad \Gamma \vdash \mathsf{true}, \mathsf{false} : \mathsf{bool} \qquad \frac{\Gamma \vdash e_i : \mathsf{bool}}{\Gamma \vdash e_1 \ \mathsf{and} \ e_2 : \mathsf{bool}} \ \lfloor\textsc{Bool}\rfloor, \lfloor\textsc{And}\rfloor$$

$$\frac{\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathtt{p} \quad \mathrm{pn}(G) \leq \mathtt{p}}{\Gamma \vdash \overline{u}[\mathtt{p}](y).P \triangleright \Delta} \ \lfloor\textsc{MCast}\rfloor \qquad \frac{\Gamma \vdash u : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, y : G \upharpoonright \mathtt{p}}{\Gamma \vdash u[\mathtt{p}](y).P \triangleright \Delta} \ \lfloor\textsc{MAcc}\rfloor$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle \Pi, e \rangle; P \triangleright \Delta, c : !\langle \Pi, S \rangle; T} \ \lfloor\textsc{Send}\rfloor \qquad \frac{\Gamma, x : S \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c?(\mathtt{q}, x); P \triangleright \Delta, c : ?(\mathtt{q}, S); T} \ \lfloor\textsc{Rcv}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T}{\Gamma \vdash c!\langle\langle \mathtt{p}, c' \rangle\rangle; P \triangleright \Delta, c : !\langle \{\mathtt{p}\}, T' \rangle; T, c' : T'} \ \lfloor\textsc{Deleg}\rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta, c : T, y : T'}{\Gamma \vdash c?((\mathtt{q}, y)); P \triangleright \Delta, c : ?(\mathtt{q}, T'); T} \ \lfloor\textsc{SRec}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta, c : T_j \quad j \in I}{\Gamma \vdash c \oplus \langle \Pi, l_j \rangle; P \triangleright \Delta, c : \oplus\langle \Pi, \{l_i : T_i\}_{i \in I} \rangle} \ \lfloor\textsc{Sel}\rfloor \qquad \frac{\Gamma \vdash P_i \triangleright \Delta, c : T_i \quad \forall i \in I}{\Gamma \vdash c\&(\mathtt{p}, \{l_i : P_i\}_{i \in I}) \triangleright \Delta, c : \&(\mathtt{p}, \{l_i : T_i\}_{i \in I})} \ \lfloor\textsc{Branch}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta'}{\Gamma \vdash P \mid Q \triangleright \Delta, \Delta'} \ \lfloor\textsc{Conc}\rfloor$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{if} \ e \ \mathsf{then} \ P \ \mathsf{else} \ Q \triangleright \Delta} \ \lfloor\textsc{If}\rfloor \qquad \frac{\Delta \ \mathsf{end\ only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta} \ \lfloor\textsc{Inact}\rfloor \qquad \frac{\Gamma, a : \langle G \rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a)P \triangleright \Delta} \ \lfloor\textsc{NRes}\rfloor$$

$$\frac{\Gamma \vdash e : S \quad \Delta \ \mathsf{end\ only}}{\Gamma, X : S\,T \vdash X\langle e, c \rangle \triangleright \Delta, c : T} \ \lfloor\textsc{Var}\rfloor \qquad \frac{\Gamma, X : S\,T, x : S \vdash P \triangleright y : T \qquad \Gamma, X : S\,T \vdash Q \triangleright \Delta}{\Gamma \vdash \mathsf{def} \ X(x, y) = P \ \mathsf{in} \ Q \triangleright \Delta} \ \lfloor\textsc{Def}\rfloor$$

Table 5: Typing rules for pure processes

$\Delta$ is the *session environment* which associates channels to session types. Formally we define:

$$\Gamma ::= \emptyset \mid \Gamma, u : S \mid \Gamma, X : S\,T \quad \text{and} \quad \Delta ::= \emptyset \mid \Delta, c : T$$

assuming that we can write $\Gamma, u : S$ only if $u$ does not occur in $\Gamma$, briefly $u \notin dom(\Gamma)$ ($dom(\Gamma)$ denotes the domain of $\Gamma$, i.e., the set of identifiers which occur in $\Gamma$). We use the same convention for $X : S\,T$ and $\Delta$ (thus we can write $\Delta, \Delta'$ only if $dom(\Delta) \cap dom(\Delta') = \emptyset$).

Table 5 presents the typing rules for pure processes.

The typing system statically ensures safety, but it is also possible to check safety properties locally with the introduction of monitors ([1]), which is an important feature in distributed networks. Indeed, while static verification has many advantages, it is usually not suitable for a real-world large-scale distributed systems: we need dynamic verification.

In [1], they introduce a light weight runtime layer, underlying the dynamic verification framework: it is called conversation runtime. It is based on the attribution of an unanmbiguous conversation identifier and a message label to message flows of a given conversation. As well as the static verification technique, the dynamic one is decentralised. Then it is possible to mix the two techniques and ensure deadlock-freedom and protocol conformance for networks formed of verified unmonitored principals and monitored ones.

## 3 Implementation Issues

We now concentrate on the implementation side. For these new theoretic concepts to be useful for developers, it must indeed be easily implementable. We present some developments that can be found in the literature.

### 3.1 Extensions of Java and C

We first expose briefly an extension of Java, called SJ, shorthand for session-based extension of Java, from [6]. It offers a full compilation-runtime framework that supports session abstraction over concrete transports, for instance TCP.
Session programming consists of two steps:

- specifying the intended interaction protocols using session types,
- implementing these protocols using session operations.

In SJ, session types are called protocols. At the application-level, the operators handle, transport-independantly, with all the expected features of a language for session-based distributed programming, ie session sockets, session server socket, session server-address, session communication (send and receive, iteration, branching), session failure and session delegation.

From this basis, the compilation-runtime framework of SJ works across three layers:

- Layer 1: SJ source code,
- Layer 2: Java translation and session runtime APIs,
- Layer 3: Runtime execution: JVM and SJ libraries.

It is the role of the SJ compiler to map layer 1 to layer 2. Besides translating the session operations into the communication primitives of the session APIs, it type-checks the source according to the constraints of both standard Java typing and session typing.

We now introduce another extension: Multiparty Session C. It defines a session runtime library and a session type checker to support a full garantee of deadlock-freedom, type-safety, communication safety and global progress for any well-typed programs. As the framework is based on theroy of multiparty session types, a session C program, ie a C program that calls the session runtime library, is developed in a top-down approach: from the global protocol, using Scribble (defined later), to the individual program. The session type-checker validates the source code against its endpoint protocol. Details of the code can be found in [7].

## 3.2 A new language: Scribble

With the ongoing endeavour to build a core descriptive framework and the associated development environment for large-scale distributed systems based on session types, there is a real need of a simple language for defining session types protocols. Indeed a protocol offers an agreement on the ways interactions proceed among two or more participants. Scribble was introduced to describe these interactions ([4]). We have seen an example of Scribble program in Figure 3.

The main elements of a Scribble protocol are: the conversation (here DataAcquisition) and the principals (here U, A and I). A conversation, or a session, is an instanciation of a protocol that follows the protocol's rules of engagement. Principals represent entities, such as corporations and individuals, who are responsible for performing communication actions in distributed applications. When a principal participates in a conversation (ie becoming its participant), it does so by taking up specific role(s) stipulated in the underlying protocol. There are two main constructs: the preamble and the protocol definition with the role declarations and interactions description.

Scribble ensures for transport asynchrony, message order preservation and reliability. The safety assurance is given by session types.

# 4 The ongoing research

In the previous sections, we have described some research work from the past few years about distributed programming. From now on, we wil focus on present and future challenges.

## 4.1 Generalised Multiparty Session Types

In recent work ([3]), a new global type syntax has been introduced. The new syntax is an extension of the previous one with join and merge constructs, which therefore explicitly distinguish the branching points from the forking ones. The goal is to build a new subclass of safe Communicating Finite State Machines, called multi session automata, which implement the local types, issued from the projection of the global type.

This global type syntax was chosen to support general graphs. Therefore it is possible to match this generalised graph syntax to a representation. Its grammar is given below:

| | | | |
|---|---|---|---|
| G | ::= | def $G$ in x | Global type |
| $G$ | ::= | x = p $\rightarrow$: $l\langle U\rangle;x'$ | Labelled messages |
| | \| | x = x' \| x" | Fork |
| | \| | x = x' + x" | Choice |
| | \| | x \| x' = x" | Join |
| | \| | x + x' = x" | Merge |
| | \| | x = end | End |
| $U$ | ::= | $\langle G\rangle$ \| bool \| $nat$ \| ... | Sorts |

We will focus on this syntax for the project, as we have seen in the motivation example: the scenario of Figure 2 combines for instance recursion, fork and choice.

A global type G ::= def $G$ in $x_0$ describes an interaction between a fixed number of participants. $x_0$ is the initial states, from where interactions start, then the state variables in $G$ specify the other interactions that can be done within this global type.

### 4.2 The tool chain

So far we have presented some work based on session types. There is now a need to link these different points altogether. The goal is to provide to developers a complete chain tool from the graph representation of the global scenario to the endpoint programs. It can be divided into three main steps:

- An equivalence between a graph representation of a choreography and the corresponding global protocol,
- The projection from the global protocol to the local protocols,
- The implementation of the endpoint programs.

Scribble is already part of this tool chain and helps with the second point. The rest is still challenging researchers and consists in future work.

## 5 Progress and Contribution

Since the beginning of March, I have done some reading work on what is described above. Following the same scheme as the one of this report, I have discovered a part of the research on the topic and understood the concepts. In parallel, I have applied this knowledge to typical examples and I have also created my own examples for the sake of the project.

From this basis, we have defined the scope of the contribution part of my project.

### 5.1 Aim of the project

As we have seen in the previous section, some work can be done on an equivalence between a graph representation of a choreography and the corresponding global protocol. Therefore my project will focus on this aspect.

We can identify several steps:

- Definition of a graph representation for global protocol as choreography,
- Development of the tool to put in relation the graph representation and the global protocol,
- Proof of the correspondence.

As far as the first step is concerned, we have first to agree on a graph representation. That is why I have begun with doing some research work on graph representation for choreography. I have also tested one on the use case presented as a motivation example.

## 5.2 BPMN Choreography

In the literature, there are already some grahic tools. We present one of them here, wiedly-used among process designers, called BPMN Choreography (BPMN stands for Business Process Modeling Notation).
BPMN 2.0 from the Object Management Group (OMG) is a standardized graphical notation for modeling business processes. It is intended to provide a notation that is readily understandable by all business users (including business analysts, technical developers, and those who will manage and monitor the processes after implementation) and to create a standardized bridge between business process design and XML-based business execution languages, such as BPEL4WS and Sybase Integration Orchestrator. BPMN 2.0 provides the following diagrams:

- Conversation diagrams - which provide an overview of the communications between participants,
- Choreography diagrams - which focus on the detail of the conversation between two or more participants, and which are often linked to specific conversation nodes,
- Collaboration diagrams - which focus on the messages that pass between participants; participants can be shown as black boxes or with processes inside them,
- Process diagrams - which focus on the sequence flow in a single process in a participant.

Choreography diagrams are used to analyze how participants exchange information to coordinate their interactions. A diagram consists in a sequence of choreographic tasks. For each task, messages can be attached: one from the initiator of the task, and possibly a reply message from the receiver. The diagram therefore shows the interactions between the participants.

## 5.3 Work in progress

I am currently working on the different possibilities for the graph representation of global types. I can either use an existing tool, for instance BPMN choreography, or develop my own tool for our purpose. The choice depends on the identification of the real need.

One advantage for BPMN Choreography is that it is the tool currently used for choreography. But it was originally meant for describing orders in tasks, rather than interactions between processes, and we need to focus on these interactions.

To help me decide this choice I focus on use cases. The first one I have developped in the one we have seen in the motivation part. As distributed networks are widely used, there are lots of concrete examples I can work with, for instance a conference management with a system like EasyChair.

### 5.4 Planed work

Here we state the different tasks we plan to achieve, as next phases of the project.

- Mid of June: reach agreement on the graph representation and the language of development, apply it to several examples,
- End of June: define the methodoly for the implementation part, prove conformance,
- July: development of the graphic tool,
- August: writing of the report.

## Acknowledgement

I thank Rumyana Neykova for her time and explanations to help me understand and define the scope of my project.

## References

[1] L. Bocchi, T.C. Chen, R. Demangeon, P.M. Deniélou, K. Honda, R. Hu, R. Neykova, and N. Yoshida. Safety assurance framework for distributed systems through multiparty session types.

[2] M. Coppo, M. Dezani-Ciancaglini, L. Bettini, and N. Yoshida. Global progress and its inference for dynamically interleaved multiparty sessions.

[3] P.M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP. LNCS*, 2012.

[4] K. Honda, A. Mukhamedov, G. Brown, T.C. Chen, and N. Yoshida. Scribbling interactions with a formal foundation. *Distributed Computing and Internet Technology*, pages 55–75, 2011.

[5] K. Honda, V. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. *Programming Languages and Systems*, pages 122–138, 1998.

[6] R. Hu, N. Yoshida, and K. Honda. Session-based distributed programming in java. *ECOOP 2008–Object-Oriented Programming*, pages 516–541, 2008.

[7] N. Ng, N. Yoshida, and K. Honda. Multiparty session c: Safe parallel programming with message optimisation.

[8] N. Yoshida and V.T. Vasconcelos. Language primitives and type discipline for structured communication-based programming revisited:< i> two systems for higher-order session communication</i>. *Electronic Notes in Theoretical Computer Science*, 171(4):73–93, 2007.