Structured Global Programming for Communication Behaviour

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

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

Abstract. This paper presents two different paradigms of description of communication behaviour, one focussing on global message flows and another on end-point behaviours, as formal calculi based on session types. The global calculus originates from Choreography Description Language, a web service description language developed by W3C WS-CDL working group. The end-point calculus is a typed π-calculus. The global calculus describes an interaction scenario from a vantage viewpoint; the endpoint calculus precisely identifies a local behaviour of each participant. After introducing the static and dynamic semantics of these two calculi, we explore a theory of endpoint projection which defines three principles for well-structured global description. The theory then defines a translation under the three principles which is sound and complete in the sense that all and only behaviours specified in the global description are realised as communications among end-point processes. Throughout the theory, underlying type structures play a fundamental role.

1 Introduction

Communication-Centred Programming. The explosive growth of Internet and World Wide Web in the last decades led to, in the form of de facto standards, an omnipresent naming scheme (URI/URL), omnipresent communication protocols (HTTP and TCP/IP) and an omnipresent message format (XML). These three elements offer the key infrastructural bases for application-level distributed programming. The software systems which make use of these and other common web standards for distributed communications are often called *web services*. Web services are an active area of infrastructural development, involving two major standardisation bodies, W3C and Oasis, and other private and public organizations.

One of the application domains which can naturally exploit the infrastructural basis of web services is the so-called business protocols. A business protocol is a series of structured and automated interactions among two or more business entities used for achieving their goals. Business protocols are inherently inter-domain, are often regulation-bound, and demand clear shared understanding about its meaning among multiple organisations with possibly conflicting interests. Numerous business protocols will be designed and implemented. Some business protocols such as industry standard will last long once specified; others would arise from temporary business needs and may undergo frequent updates. Because of its inherent inter-organizational nature, there is a strong demand for a common standard for specifying well-founded and correct business protocols.

Global Description of Interaction. One of the standardisation efforts for a language to specify business protocols is the Web Services Choreography Description Language (WS-CDL) [40], developed by W3C's WS-CDL Working Group since 2004 in collaboration with π-calculus experts as scientific advisors. WS-CDL is a specification language which directly describes global information flows and their structures, close to, for example, the standard notation for cryptographic protocols [26], UML sequence diagrams [27] and message sequence charts (MSC) [21]. Unlike these predecessors, in order to enable precise description and specification of complex business protocols, WS-CDL offers a fully expressive description language for channel based communication, equipped with standard control constructs (e.g. sequencing, conditionals and loops) and is conceived with potential for type-based and other formal validation. The underlying intuition behind *choreography* can be summarised as follows.

> *"Dancers dance following a global scenario without a single point of control."*

WS-CDL is a language for describing such a "global scenario" for business protocols. The description can then be executed by individual distributed processes without a single point of control.[1] Another significant feature of WS-CDL is its use of *sessions* for organizing communication behaviour: at the outset of each unit of a business protocol, a session is established between each pair of communication parties so that involved communications can be distinguished from any other instances of business protocols.

Endpoint Projection. A global description of communication behaviour arguably offers conceptual clarity not found in endpoint-based descriptions, partly because a global interaction flow *is* often the central objective a communication-based application is intended to realise. Real execution of the description, however, is always through communication among endpoints which (as the notion of choreography dictates) may as well involve no centralised control. Thus we ask:

> *How can we project a global description to endpoint processes so that their interactions precisely realise the original global description?*

Such a projection may be called *endpoint projection (EPP)*, the term originating from WS-CDL WG.

What are criteria for a good EPP? We naturally desire an EPP to be *sound* and *complete*, in the sense that all and only globally described behaviour is realised as communications among endpoints. We may regard such an EPP as giving the semantics of a global description.

An appropriate notion of EPP leads to significant engineering usage of a global description:

1. (code generation) For a global description with full algorithmic details, we can create a (perhaps multi-language) *complete distributed application* by projecting it to each of its endpoints.

[1] An contrasting idea in web service is *orchestration* where one master component, "conductor", directly controls activity of one or more slave components, which is useful in the intra-organisational applications.

2. (prototype generation) Projection can also be used for generating a *skeleton code* for each endpoint which only contains basic communication behaviour, to be elaborated to full code.
3. (conformance) A team of programmers initially agree on a shared global specification for interactions among endpoints: during/after programming, each programmer can check if her/his code conforms to the specification by conformance checking against projection. This scheme also applies to conformance of existing services/libraries to a given scenario.
4. (runtime monitoring, testing and debugging) At runtime, each endpoint can check if ongoing communications at his/her site conform to the global description by checking against its projection to that endpoint. The monitoring can also be used for debugging and testing existing code.
5. (property validation) Various static analyses/logical validation can be done for a global description so that they make sense for each endpoint through EPP.

Many of these ideas come from discussions in WS-CDL working group and are partly already realised in an open-source reference implementation of WS-CDL [28]. For example, runtime monitoring is a basic expected use of WS-CDL with relevance to regulatory concerns, especially for financial protocols. For all of these uses, EPP should be built on a clear, precise understanding of semantics of global and local descriptions, guaranteeing exact match between them.

This Work. The present paper introduces two typed calculi for interaction, one being a distillation of WS-CDL and another an applied version of the π-calculus, and develops a theory of endpoint projection. Our central contribution is the identification of natural descriptive principles for global descriptions which induce a type-preserving EPP that is sound and complete with respect to their operational semantics. There are three principles:

- *Connectedness*, a basic local causality principle.
- *Well-threadedness*, a stronger locality principle based on session types [10, 14, 16, 18, 37, 39].
- *Coherence*, a consistency principle for description of each participant in a global description.

These principles are stipulated incrementally on the basis of well-typedness. They not only enunciate natural disciplines for well-structured global description, but also offer gradually deeper analysis of global descriptions. The EPP has the following shape:

$$I \quad \mapsto \quad A[P] \mid B[Q] \mid C[R] \mid \cdots$$

where I is a global description, A, B and C are *participants* to the protocol and P, Q and R are projections of I onto A, B and C respectively. When applied to well-structured interactions, the mapping thus defined satisfies the following three properties:

- *Type preservation:* the typing is preserved through EPP.
- *Soundness:* nothing but behaviours (reductions) in I are in the image of its EPP.
- *Completeness:* all behaviours in I are in the image of its EPP.

The EPP theory is intended as a theoretical basis of global description languages including, but not limited to, WS-CDL. The theory opens a conduit between global descriptions and accumulated studies on process calculi, allowing the exploitation of rich theories for engineering concerns. A version of EPP theory will be published as an associated document of WS-CDL 1.0, and will form a part of an open-source implementation of WS-CDL [28].

Related Works. As far as we know, this work is the first to present the typed calculus based on global description of communication behaviour, integrated with the theory of endpoint projection. Global methods for describing communication behaviour have been practiced in several different engineering scenes in addition to WS-CDL (for which this work is intended to serve as its theoretical underpinning). Representative examples include the standard notation for cryptographic protocols [26], message sequence charts (MSC) [21], and UML sequence diagrams [27]. These notations are intended to offer a useful aid at the design/specification stage, and do not offer full-fledged programming language, lacking in e.g. standard control structures and/or value passing. Petri-nets [38] may also be viewed as offering a global description, though again they are more useful as a specification/analytical tool.

DiCons (which stands for "Distributed Consensus"), which is independently conceived and predates WS-CDL, is a notation for global description and programming of Internet applications introduced and studied by Baeton and others [4]. DiCons chooses to use programming primitives close to user's experience in the web, such as web server invocation, email, and web form filing, rather than general communication primitives. Its semantics is given by either MSCs or direct operational semantics. DiCons does not use session types or other channel-based typing. An analogue of the theory of endpoint projection has not been developed in the context of DiCons.

The present work shares with many recent works its direction towards well-structured communication-centred programming using types. Pict [31] is the programming language based on the π-calculus, with rich type disciplines including linear and polymorphic types (which come from the studies on types for the π-calculus discussed in the next paragraph). Polyphonic C$\sharp$ [5] uses a type discipline for safe and sophisticated object synchronisation. Compagnoni, Dezani, Gay, Vasconcelos and others have studied interplay of session type disciplines with different programming constructs and program properties [10, 14, 16, 18, 37, 39]. The EPP theory offers a passage through which these studies (all based on endpoint languages and calculi) can be reflected onto global descriptions, as we have demonstrated for session types in the present work. In the context of session types, the present work extends the session structure with multiple session names which is useful for having parallel communications inside a session.

Many theories of types for the π-calculus are studied. In addition to the study of session types mentioned above, these include input/output types [24, 30], linear types [17, 22], various kinds of behavioural types [3, 6, 7, 19, 20, 35, 36, 41] and combination of behavioural types and model checking for advanced behavioural analysis [32, 33], to name a few. Among others, behavioural types offer an advanced analyses for such phenomena as deadlock freedom. We are currently studying how these advanced type-based validationon techniques on the basis of the present simple session type discipline

will lead to effective validation techniques. Again these theories would become applicable to global descritpions through the link established by the EPP theory.

Gordon, Fournet, Bhargavan and Corin studied security-related aspects of web services in their series of works (whose origin lies in the security-enhanced pi-calculus called spi-calculus [2]). In their recent work [8], the authors have implemented part of WS-Security libraries using a dialect of ML, and have shown how annotated application-level usage of these security libraries in web services can be analysed with respect to their security properties by translation into the π-calculus [9]. The benefits of such a tool can be reflected onto the global descriptions through the theory of EPP, by applying the tool to projections.

Laneve and Padovani [23] give a model of orchestrations of web services using an extensions of π-calculus to join patterns. They propose a typing system for guaranteeing a notion of smoothness i.e. a constraint on input join patterns such that their subjects (channels) are co-located in order to avoid a classical global consensus problem during communication. Reflecting the centralised nature of orchestration (cf. footnote 1), neither a global calculus nor endpoint projection is considered. A bisimulation-based correspondence between choreography and orchestration in the context of web services has been studied in [11] by Busi and others, where a notion of state variables is used in the semantics of the orchestration model. They operationally relate choreographies to orchestration. Neither strong type systems nor disciplines for end-point projection are studied in their work.

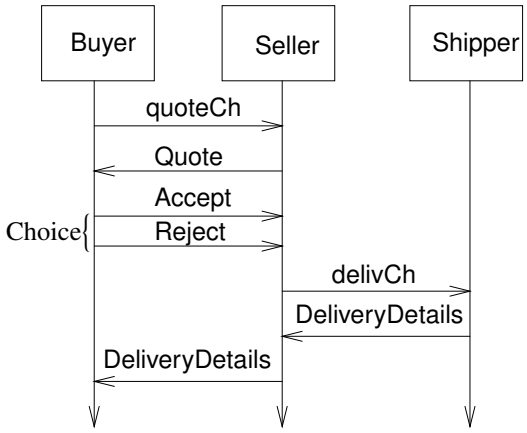
Outline. Section 2 previews the key technical ideas using concrete examples. Sections 3 and 4 outline the global and endpoint calculi, introducing their static and dynamic semantics. Section 5 develops the theory of endpoint projection. Section 6 summarises further results and applications of end-point projection. Section 7 concludes with further topics. The appendix presents a larger example illustrating how the endpoint projection concretely works. Many examples and the full technical developments of the theory are found in the full versions [12, 13].

Acknowledgements. The present work is part of ongoing collaboration between W3C WS-CDL working group and a team of π-calculus experts, led by Robin Milner. Its development has benefitted from the extensive discussions with WS-CDL working group members. In particular we thank Steve Ross-Talbot and Gary Brown for our many fascinating (and ongoing) conversations.

2 Preview of Key Technical Ideas

2.1 Buyer-Seller Protocol.

This section gives an outline of key technical ideas. Throughout we consider a simple business protocol from [34], which we call "Buyer-Seller Protocol", and its variations. In the core protocol, the participants involved are a Buyer, a Seller and a Shipper. We first describe the protocol in the following sequence diagram.



In words, the protocol consists of the following actions.

(1) Buyer asks Seller, through a specified channel, to offer a quote (denoted quote) for buying a specific good;
(2) Seller replies with a quote;
(3) Buyer then answers with either an accept or a reject.
(4) *In the case of acceptance,*
  (4-1) Seller sends the order to the Shipper;
  (4-2) Shipper sends the delivery details back to the Seller.
  (4-3) Seller forwards them to Buyer: the protocol terminates.
(5) *In the case of rejection*, the protocol terminates.

Note the diagram is ambiguous on the branching Actions (4) and (5): the purpose of such diagrams is to offer an informal overview rather than precise specification. Nevertheless, protocols tend to be complex, with nondeterministic and conditional choices, loops, timeout and other elements. This motivates a need of a *syntactic means*, i.e. a language, for describing such protocols.

Some of the central elements of such a language (WS-CDL is one example) may be determined by observing that the whole intention of specifying such protocols is to *instantiate it repeatedly*, including its shared usage (e.g. the Buyer role can be assumed by any potential and possibly concurrent buyers). This consideration leads to the following two simple engineering principles.

Service Channel Principle (SCP): Invocation channels (e.g. a channel at which Buyer first communicates to Seller, similarly Seller to Shipper) can be shared and invoked repeatedly.

Session Principle (SP): A sequence of conversations belonging to a protocol should not be confused with other concurrent runs of this or other protocols by the participants: in other words, each such sequence should form one logical unit of a conversation, or a *session*.

(**SCP**) does not preclude a channel is only known to a closed number of participants. It corresponds to a replicated channel in the π-calculus, or, more accurately, a replicated

channel which is not prefixed by other input prefixes (such channels are called *uniformly receptive* in [35] and *server channels* in [6]. (**SP**) can have complex forms, but a most basic one is a dyadic one which allows simple and robust type abstraction with tractable type checking [14, 18, 39]. [2] These two principles are central for the whole technical development in the paper.

2.2 A Language for Global Description of Communication.

The following presents the full description of Buyer-Seller Protocol in the *global calculus*, whose syntax and semantics we shall formally introduce later.

1. Buyer $\rightarrow$ Seller : quoteCh($\boldsymbol{\nu}\, s$)**.**
2. Seller$\rightarrow$Buyer : $s\langle$Quote, 300, $x\rangle$**.** {
3. {Buyer$\rightarrow$Seller : $s\langle$Accept$\rangle$**.**
4. Seller $\rightarrow$ Shipper : delivCh($\boldsymbol{\nu}\, t$)**.**
5. Shipper$\rightarrow$Seller : $t\langle$DelivDetails, $v_{\texttt{details}}$, $x_{\texttt{details}}\rangle$**.**
6. Seller$\rightarrow$Buyer : $s\langle$DelivDetails, $x_{\texttt{details}}$, $y_{\texttt{details}}\rangle$**. 0** }
7. +
8. {Buyer$\rightarrow$Seller : $s\langle$Reject$\rangle$**. 0**}
9. }

Line 1 describes Action (1) in the previous informal description of the protocol. The quoteCh is a *service channel*, which may be considered as a publicly known URL for a specific service. The invocation marks the start of a session between the buyer and the seller: the $\boldsymbol{\nu}$-bound s is a *session name*, a fresh name that will be used for later communication in this session. Unlike standard process calculi, the syntax no longer describes input and output actions separately: the information exchange is directly described.

Line 2 describes Action (2) in the scenario, Seller's reply to Buyer. The session has already been started and now the two participants communicate using the session name s. In addition, three factors involved: Quote identifies the particular operation used in this communication (i.e. request of quote), 300 is the quote sent by Seller; x is a variable located at Buyer where the communicated value will be stored.

Lines 3/8 describe Action (3), where Buyer communicates its choice (Accept or Reject) to Seller through s. Two series of actions which follow these choices are combined by + in Line 7. If Accept is chosen, Seller sends Shipper the Buyer's details via the service channel delivCh of Shipper, creating a fresh session name t (Line 4). Then in Line 5, Shipper sends back the shipping details through t. Finally Seller forwards the details to Buyer in Line 6, where the protocol terminates. In Line 8, Buyer communicates Reject, in which case the protocol immediately terminates.

The code above offers a precise global description of the informal scenario above, drawing on (**SCP**) and (**SP**). Sessions offer logical grouping of threads of interactions,

[2] In implementations of web services, sessions are implemented using so-called *co-relation identities* (which may be considered as nonces in cryptographic protocols). This and the channel-based representation usually employed in the study of session types are logically equivalent, as discussed in [2].

where each thread starts with a procedure-call-like service invocation. This last feature can be seen more clearly in the following refinement of the code above.

```
1.  Buyer → Seller : quoteCh(ν s).
2.  rec X. {
3.  Seller → Buyer : s⟨Quote, q, x⟩.
4.  if reasonable(x)@Buyer then
5.     {Buyer → Seller : s⟨Accept⟩.
6.       Seller → Shipper : delivCh(ν t).
7.       Shipper → Seller : t⟨DelivDetails, v_details, x_details⟩.
8.       Seller → Buyer : s⟨DelivDetails, x_details, y_details⟩. 0 }
9.  else
10.    {Buyer → Seller : s⟨Reject⟩. q := q − 1@Seller. X } }
11. }
```

Above if Buyer chooses Reject, the protocol recurs to Line 3 after decrementing the quote. In Line 4, we assume a unary predicate $\texttt{reasonable}(x)$ evaluated at Seller ("@" indicates a location, similarly in Line 10). Note the session notation makes it clear that all Quote-messages from Seller to Buyer in the recursion are done within a single session. Later in this preview we shall present another example where such session information plays a crucial role in tractable endpoint projection.

For comparison we present the endpoint counterpart of the first simple global code. The first is the endpoint code of Buyer.

$$\begin{aligned}&\mathsf{Buyer}[\ \overline{\mathsf{QuoteCh}}(\nu s).\ s \rhd \mathsf{Quote}(x).\ \{\\ &\qquad \{\, s \lhd \mathsf{Accept}.\ s \rhd \mathsf{DeliveryDetails}(y_{\texttt{details}}).\ \mathbf{0}\,\} +\\ &\qquad \{\, s \lhd \mathsf{Reject}.\ \mathbf{0}\}\ \}\quad]\end{aligned}$$

Above Buyer[P] indicates a participant (a named agent) whose behaviour is given by P. The Seller's code is given as:

$$\begin{aligned}&\mathsf{Seller}[\ !\,\mathsf{QuoteCh}(s).\ \overline{s} \lhd \mathsf{Quote}\langle 300\rangle.\ \{\\ &\qquad \{\, s \rhd \mathsf{Accept}.\\ &\qquad\qquad \overline{\mathsf{DeliveryCh}}(\nu t).\ t \rhd \mathsf{DeliveryDetails}(x_{\mathsf{details}}).\\ &\qquad \overline{s} \lhd \mathsf{DeliveryDetails}\langle x_{\texttt{details}}\rangle.\ \mathbf{0}\,\}\quad +\\ &\qquad \{\,\mathsf{Reject}.\ \mathbf{0}\,\}\ \}\quad]\end{aligned}$$

The code of Shipper is similar. Observe endpoint descriptions clearly depict local communication behaviour. However they do not directly describe how interaction proceeds globally, which may often be the central concern of the designers and users of a communication-centred application. The two service channels (*QuoteCh* and *DeliverCh*) are replicated and ready to receive invocations, following (**SCP**).

As may be seen above, extraction of behaviour from a global description relies on session information. We illustrate this point further. Consider the following snippet of

global description, where a and b are used to indicate the lack of session information.

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Seller} : a\langle \mathsf{QuoteReq},\ pname1,\ pname1\rangle\mathbf{.}\\
\mathsf{Seller} \to \mathsf{Buyer} : b\langle \mathsf{QuoteRes},\ quote1,\ quote1\rangle\mathbf{.}\\
\mathsf{Buyer} \to \mathsf{Seller} : a\langle \mathsf{QuoteReq},\ pname2,\ pname2\rangle\mathbf{.}\\
\mathsf{Seller} \to \mathsf{Buyer} : b\langle \mathsf{QuoteRes},\ quote2,\ quote2\rangle\mathbf{.}\ I
\end{array}$$

Here Buyer requests a quote twice: it may look that the behaviour of Seller is such that it allows a consecutive quote requests in one go. This ambuity is resolved if we put a session information:

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Seller} : ch(s)\langle \mathsf{QuoteReq},\ pname1,\ pname1\rangle\mathbf{.}\\
\mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{QuoteRes},\ quote1,\ quote1\rangle\mathbf{.}\\
\mathsf{Buyer} \to \mathsf{Seller} : ch(t)\langle \mathsf{QuoteReq},\ pname2,\ pname2\rangle\mathbf{.}\\
\mathsf{Seller} \to \mathsf{Buyer} : t\langle \mathsf{QuoteRes},\ quote2,\ quote2\rangle\mathbf{.}\ I
\end{array} \qquad (1)$$

(Above we use a construct which combines a session initiation and an in-session communication. This is convenient for practice: our theoretical treatment in the present paper separates these two for clearer formal presentation, with no loss of generality via a simple encoding.) Using the session information, we infer:

$$!ch(s)[\langle \mathsf{QuoteReq}\rangle(pname).\overline{s}\langle \mathsf{QuoteRes}\rangle(quote).P]$$

Note the endpoint behaviour would have been quite different if we represent all request-replies as belonging to a single session.

2.3 Disciplines for Global Description.

Even if a global flow of interaction is the primary concern of an application designer, in implementation, a global scenario has to be realised by distributed end-points communicating with each other. Thus we need to bridge the world of global description to endpoint descriptions. Our ultimate aim is to have global descriptions such that their operational content, or endpoint realisation, is transparent from these descriptions.

Having such a bridge is non-trivial because a global calculus allows *description of communication behaviour that does not make sense at endpoints.* As the first such issue, let us consider the following code snippet for global description:

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s)\mathbf{.}\\
\mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t)\mathbf{.}\ \mathbf{0}
\end{array}$$

Above Shipper is supposed to contact Depot only after Buyer performed a request to Seller. Implementing such a system demands Shipper is notified once the initial communication is performed, i.e. there is an implicit communication from Seller to Shipper:

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s)\mathbf{.}\\
\mathsf{Seller} \to \mathsf{Shipper} : ch(\boldsymbol{\nu}\, s')\mathbf{.}\\
\mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t)\mathbf{.}\ \mathbf{0}
\end{array}$$

With this insertion, the description is realisable purely through explicitly specified message exchanges. The criteria which says each participant acts only as a result of its local event (such as reception of a message) is called *connectedness*. We shall give its formal definition in Section 5.

Connectedness is an intuitive idea for well-structured global description. The next condition is more subtle. Consider the following (connected) interaction:

$$\begin{array}{l}\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s)\mathbf{.}\\ \mathsf{Seller} \to \mathsf{Shipper} : ch_2(\boldsymbol{\nu}\, t)\mathbf{.}\\ \mathsf{Shipper} \to \mathsf{Buyer} : ch_3(\boldsymbol{\nu}\, u)\mathbf{.}\\ \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{op},\ v,\ x\rangle\mathbf{.}\ I\end{array}$$

Above we assume Buyer offers a service channel ch_3 which is useful for Shipper. We claim that this global code (regardless of ensuing interaction at I) is unrealisable at endpoints, at least under the natural type discipline and code organisation.

The first action tells that there is a thread in Buyer which invokes Seller. This thread becomes inactive in the second line. In the third line, a service at ch_3 in Buyer is invoked. In the final line, Buyer communicates to Seller via a session name s opened in the first action. So, at the endpoint, we should have the following two chunks of the code:

$$\overline{ch_1}(\boldsymbol{\nu} s)\mathbf{.}\ \overline{s} \lhd \mathsf{op}\langle v\rangle.P \ \mid\ \mathbf{!}\, ch_3(t).Q$$

The first chunk is for the initial invocation and ensuring reception of op in the same session, while the second is a service at ch_3 (by (**SCP**) this channel should be ready to receive invocations). Note that, by $\overline{s} \lhd \mathsf{op}\langle v\rangle$ belonging to a session s, this action cannot be located under ch_3. On the other hand, the code of Seller should be:

$$\mathbf{!}\, ch_1(s)\mathbf{.}\ \overline{ch_2}(\boldsymbol{\nu} t)\mathbf{.}\ t \rhd \mathsf{op}(x)$$

Finally, the code of Shipper becomes:

$$\mathbf{!}\, ch_2(t)\mathbf{.}\ \overline{ch_3}(\boldsymbol{\nu} u).R$$

We can now have the three endpoint processes get engaged in communications: First, Buyer invokes ch_1, then Seller invokes ch_2 of Shipper: up to here the interaction follows the original global scenario. However, at this point, the action $s \rhd \mathsf{op}(x)$ is free to react with its dual action $\overline{s} \lhd \mathsf{op}\langle v\rangle$, before Shipper invokes Seller's the other component, the service at ch_3. Thus the sequencing in the global description can be violated.

The fundamental issue here is that the given global code assumes a false (unrealisable) dependency among actions: the last action belongs to a thread which started from the invocation of ch_1, while the description says it should take place as the direct result of the third action at a distinct thread which has been opened by the invocation at ch_3. If a global description is free from such false dependency, we say it is *well-threaded*. We shall see in Section 5 that checking well-threadedness is simple and mechanical.

Well-threadedness not only eliminates false dependency but also allows consistent extraction of threads (i.e. sequences of actions) from a given global code. These threads

become the constituents of endpoint processes in EPP. The final well-structuring principle is concerned with this composition. It is often necessary to *merge* threads to obtain the final endpoint behaviour of a single service.[3] Consider the following parallel composition:

$$\mathsf{Buyer} \to \mathsf{Seller} : ch(\nu\, s).\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{op}_1,\ e,\ x_1\rangle.\ I_1 \ \mid$$
$$\mathsf{Buyer} \to \mathsf{Seller} : ch(\nu\, t).\ \mathsf{Seller} \to \mathsf{Buyer} : t\langle \mathsf{op}_2,\ e,\ x_2\rangle.\ I_2$$

where $op_1 \neq op_2$. Here Buyer invokes Seller's service at ch twice in parallel. Now consider constructing the code for this service at channel ch: then we need to merge these two threads into one endpoint behaviour. But the global description is contradictory, since in one invocation the service reacts with op_1, while in another the service reacts with op_2. The description is not self-consistent.

A central issue is that, in a global description, the descriptions of a single endpoint behaviour (especially a service at a service channel) can be *scattered in different portions of the code*. Thus, without these scattered descriptions being consistent with each other, we cannot merge them into a single behaviour. We call such mergeability, *coherence*: coherence is not simply about identity of the behaviour, as in the above case, since distinct input branches may be described in different portions of a global code. The details are given in Section 5. Coherence can again be checked mechanically.

With coherence as the final well-structuredness condition, we can now project a global code to endpoint behaviours that precisely realise the original global scenario.

3 The Gobal Calculus

3.1 Syntax

The syntax of the global calculus is given by the following BNF. $I, I', \ldots$ denote *terms* of the calculus, also called *interactions*. Below $ch, ch' \ldots$ range over *service channels*, intuitively denoting the shared channels of (web) services; $s, t, \ldots$ range over *session names*; $\tilde{s}$ indicates a vector of session names; $A, B, C, \ldots$ range over *participants*; $x, y, z, \ldots$ over local variables in each participant; $X, X', \ldots$ over *term variables*; and $e, e', \ldots$ over arithmetic and other first-order expressions.

$$\begin{array}{rll}
I ::= & A \to B : ch(\nu\, \tilde{s}).\ I & \text{(init)} \\
\mid & A \to B : s\langle \mathsf{op},\ e,\ y\rangle.\ I & \text{(comm)} \\
\mid & x@A := e.\ I & \text{(assign)} \\
\mid & I_1 \mid I_2 & \text{(par)} \\
\mid & \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 & \text{(ifthenelse)} \\
\mid & I_1 + I_2 & \text{(sum)} \\
\mid & (\nu s)\ I & \text{(new)} \\
\mid & X & \text{(recvar)} \\
\mid & \mu X.\ I & \text{(rec)} \\
\mid & \mathbf{0} & \text{(inaction)}
\end{array}$$

[3] This merging already takes place in the extraction of code in (1) above, though in a trivial way.

Table 1 Reduction Relation for the Global Calculus

$$\text{(G-Init)}\ \frac{-}{(\sigma,\ A \to B : ch(\boldsymbol{\nu}\, \tilde{s}).\ I) \to (\sigma, (\boldsymbol{\nu}\tilde{s})\ I)} \qquad \text{(G-Com)}\ \frac{\sigma' = \sigma[x@B \mapsto v] \quad \sigma \vdash e@A \Downarrow v}{(\sigma,\ A \to B : s\langle \mathsf{op},\ e,\ x\rangle.\ I) \to (\sigma',\ I)}$$

$$\text{(G-Asgn)}\ \frac{\sigma' = \sigma[x@A \mapsto v] \quad \sigma \vdash e@A \Downarrow v}{(\sigma, x@A := e.\ I) \to (\sigma',\ I)} \qquad \text{(G-Sum)}\ \frac{i = 1, 2}{(\sigma,\ I_1 + I_2) \to (\sigma',\ I_i)}$$

$$\text{(G-IfT)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{tt}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \to (\sigma, I_1)} \qquad \text{(G-IfF)}\ \frac{\sigma \vdash e@A \Downarrow \mathsf{ff}}{(\sigma,\ \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2) \to (\sigma,\ I_2)}$$

$$\text{(G-Par)}\ \frac{(\sigma,\ I_1) \to (\sigma',\ I_1')}{(\sigma, I_1 \mid I_2) \to (\sigma',\ I_1' \mid I_2)} \qquad \text{(G-Res)}\ \frac{(\sigma,\ I) \to (\sigma',\ I')}{(\sigma, (\boldsymbol{\nu}\tilde{s})\ I) \to (\sigma',\ (\boldsymbol{\nu}\tilde{s})\ I')}$$

$$\text{(G-Rec)}\ \frac{(\sigma,\ I[\mu X.I/X]) \to (\sigma',\ I')}{(\sigma,\ \mu X.I) \to (\sigma',\ I')} \qquad \text{(G-Struct)}\ \frac{I \equiv I'' \quad (\sigma,\ I) \to (\sigma,\ I') \quad I' \equiv I'''}{(\sigma,\ I'') \to (\sigma',\ I''')}$$

The term (init) denotes a session initiation by A to B on service channel ch with fresh session channels $\tilde{s}$ and continuation I. The interaction (com) is the in-session communication over a session channel s. Note that y is free and does not bind I. The operators $\mid$ and + denote respectively parallel and choice operators. $(\boldsymbol{\nu}s)\ I$ is the π-calculus-like name restriction, binding s in I. (ifthenelse) and (assign) are the standard conditional and assignment ($e@A$ indicates e is located at A). $\mu X.\ I$ is recursion, where the variable X is bound in I. $\mathbf{0}$ denotes termination. The free and bound session channels and term variables are defined in the usual way.

The presented syntax is intended as the minimum one for presenting examples and for the EPP theory. Section 6 discusses natural additional syntactic constructs.

3.2 Dynamics

The dynamics of the global calculus is given by reduction relation close to that of imperative languages. A *state* σ assigns a value to the variables located at each participant. We shall write $\sigma@A$ to denote the portion of σ local to A, and $\sigma[y@A \mapsto v]$ to denote a new state which is identical with σ except that $\sigma'@A(y)$ is equal to v. The reduction is the binary relation $\to$ generated by the rules in Table 1. "$(\sigma,\ I) \to (\sigma',\ I')$" says that I in the state σ performs one-step computation and becomes I' with the new state σ'.

Rule (Init) is for session initiation: after A initiates a session with B on service channel ch, A and B share $\tilde{s}$ locally (indicated by binding $(\boldsymbol{\nu}\tilde{s})$), and the next I is unfolded. The initiation channel ch will play an important role for typing and the endpoint projection later. (Com) is a key rule: the expression e is evaluated into v in the A-portion of the state σ and then assigned to the variable x located at B resulting in new the state $\sigma[x@B \mapsto v]$. Note that the same variable (say x) can be located at different participants, so that $\sigma@A(x)$ and $\sigma@B(x)$ are distinct. Similarly to the session initiation, the session channel s is attached. The rule (Struct) makes use of structural congruence. The structural congruence relation is the least congruence relation $\equiv$ on I

Table 2 Typing Rules for Global Calculus

$$\text{(G-TInit)}\ \frac{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash I \rhd \Delta \cdot \tilde{s}\,[B,A]:\alpha}{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \rhd \Delta}$$

$$\text{(G-TCom)}\ \frac{\Gamma \vdash I \rhd \Delta \cdot \tilde{s}\,[A,B]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma\ \vdash\ A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle.\ I\ \rhd\ \Delta \cdot \tilde{s}\,[A,B]:s \blacktriangleleft \Sigma_{i\in J}\mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$\text{(G-TComInv)}\ \frac{\Gamma \vdash I \rhd \Delta \cdot \tilde{s}\,[B,A]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma\ \vdash\ A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle.\ I\ \rhd\ \Delta \cdot \tilde{s}\,[B,A]:s \blacktriangleright \Sigma_{i\in J}\mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$\text{(G-TAsgn)}\ \frac{\Gamma \vdash x@A:\theta \quad \Gamma \vdash e@A:\theta \quad \Gamma \vdash I \rhd \Delta}{\Gamma \vdash x := e@A.\ I \rhd \Delta}$$

$$\text{(G-TSum)}\ \frac{\Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash I_1 + I_2 \rhd \Delta} \qquad \text{(G-TIf)}\ \frac{\Gamma \vdash e@A:\mathtt{bool} \quad \Gamma \vdash I_1 \rhd \Delta \quad \Gamma \vdash I_2 \rhd \Delta}{\Gamma \vdash \mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2 \rhd \Delta}$$

$$\text{(G-TPar)}\ \frac{\Gamma \vdash I_1 \rhd \Delta_1 \quad \Gamma \vdash I_2 \rhd \Delta_2}{\Gamma \vdash I_1 \mid I_2 \rhd \Delta_1 \bullet \Delta_2} \qquad \text{(G-TRes1)}\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2[A,B]:\alpha}{\Gamma \vdash (\boldsymbol{\nu} s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2 : \bot}$$

$$\text{(G-TRes2)}\ \frac{\Gamma \vdash I \rhd \Delta,\ \tilde{s}_1 s \tilde{s}_2 : \bot}{\Gamma \vdash (\boldsymbol{\nu} s)\ I \rhd \Delta,\ \tilde{s}_1\tilde{s}_2 : \bot} \qquad \text{(G-TRes3)}\ \frac{\Gamma \vdash I \rhd \Delta,\ \varepsilon : \bot}{\Gamma \vdash (\boldsymbol{\nu} s)\ I \rhd \Delta}$$

$$\text{(G-TRec)}\ \frac{\Gamma \cdot X:\Delta \vdash I \rhd \Delta}{\Gamma \vdash \mu X.\ I \rhd \Delta} \qquad \text{(G-TVar)}\ \frac{\Gamma,\ X:\Delta \text{ well-formed}}{\Gamma,\ X:\Delta \vdash X \rhd \Delta}$$

$$\text{(G-TZero)}\ \frac{\Gamma \text{ well-formed} \quad \forall i \neq j.\ \{\tilde{s}_i\} \cap \{\tilde{s}_j\} = \emptyset}{\Gamma \vdash \mathbf{0} \rhd \bigcup_i \tilde{s}_i[A_i,B_i]\mathsf{end}}$$

such that $\mid$ and $+$ are commutative monoids and such that it satisfies alpha-conversion and the rule $(\boldsymbol{\nu} s)\ I_1 \mid I_2\ \equiv\ (\boldsymbol{\nu} s)\ (I_1 \mid I_2)$ for $s \notin \mathsf{fn}(I_2)$.

Consider, for instance, the interaction

$$\begin{aligned}&\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{QuoteCh}(\boldsymbol{\nu}\, s).\\ &\quad \mathsf{Seller} \to \mathsf{Buyer} :\ s\langle \mathsf{Quote},\ 300,\ x\rangle.\ I'\end{aligned}$$

and let us evaluate it in the state σ. By applying rule (Init), we get the pair $(\sigma, (\boldsymbol{\nu} s)\ \mathsf{Seller} \to \mathsf{Buyer} :\ s\langle \mathsf{Quote},\ 300,\ x\rangle.\ I')$. Now, by applying rules (Res) and (Com) together in the state σ we get the pair $(\sigma[x@\mathsf{Buyer} \mapsto 300], (\boldsymbol{\nu} s)\ I')$.

3.3 Typing

We use a generalisation of session types [18] as the type discipline for the global calculus. The grammar of types follows.

$$\begin{aligned}\alpha \quad ::= \quad & s \blacktriangleright \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i \quad \mid \quad s \blacktriangleleft \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i\\ \mid\ & \alpha \mid \alpha \quad \mid \quad \mathsf{end} \quad \mid \quad \mu \mathbf{t}.\ \alpha \quad \mid \quad \mathbf{t}\end{aligned}$$

where $\theta, \theta', \ldots$ range over *value types*. $\alpha, \alpha', \ldots$ are *session types*. $s \blacktriangleright \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i$ is a *branching input type* at session channel s, indicating possibilities for receiving any of the operators from $\mathsf{op_i}$ (which are pairwise distinct) with a value of type θ_i; $s \blacktriangleleft \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i$, a *branching output type* at s, is the exact dual of the above. The type $\alpha_1 \mid \alpha_2$ is a *parallel composition of* α_1 *and* α_2, abstracting parallel composition of two sessions. We take $\mid$ to be commutative and associative, with end, the *inaction type* indicating session termination, being the identity. We demand session channels in α_1 and those in α_2 to be disjoint: this will guarantee a linear use of session channels. $\mathbf{t}$ is a *type variable*, while $\mu\mathbf{t}.\alpha$ is a *recursive type*, where $\mu\mathbf{t}$ binds free occurrences of $\mathbf{t}$ in α. In recursive types, we assume each recursion is guarded, i.e., in $\mu\mathbf{t}.\alpha$, α is an n-ary parallel composition of input/output types. Recursive types are regarded as regular trees in the standard way [29].

Note that session channels occur free in session types: this is necessary to allow multiple session channels to appear in a single session in parallel. Thus, we can faithfully capture the behaviour of web services where it is possible to exchange different data simultaneously, leading to a generalisation of session types in the literature. Let us show a simple example:

$$s \blacktriangleleft \mathsf{Quote}(\mathtt{int}).\ \mathsf{end} \mid s' \blacktriangleleft \mathsf{Extra}(\mathtt{String}).\ \mathsf{end}$$

Here a participant is sending a quote (integer) at s and extra information about the product at s' in a single session: without using distinct session channels, two communications can get confused.

The duality for session types plays the key role to guarantee dyadic interaction [18]. The *co-type*, or *dual*, of α, written $\overline{\alpha}$, is given as follows.

$$\overline{s \blacktriangleleft \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i} = s \blacktriangleright \Sigma_i \mathsf{op_i}(\theta_i).\ \overline{\alpha}_i$$
$$\overline{s \blacktriangleright \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i} = s \blacktriangleleft \Sigma_i \mathsf{op_i}(\theta_i).\ \overline{\alpha}_i$$
$$\overline{\mu\mathbf{t}.\ \alpha} = \mu\mathbf{t}.\ \overline{\alpha} \qquad \overline{\mathbf{t}} = \mathbf{t} \qquad \overline{\mathsf{end}} = \mathsf{end}$$

For instance, the co-type of $s \blacktriangleright \mathsf{Quote}(\mathtt{int}).\ \mathsf{end}$ is $s \blacktriangleleft \mathsf{Quote}(\mathtt{int}).\ \mathsf{end}$, exchanging input and output.

Each time a session is initiated via a service channel, session channels are freshly generated. Thus, the interface of a service should indicate a vector of session names to be exchanged, in addition to how they are used. This is represented by a *service type*, in which concrete instances of session names in a session type are abstracted, written: $(\tilde{s})\alpha$ where $\tilde{s}$ is a vector of pairwise distinct session channels covering all session channels in α, and α does not contain free type variables. $(\tilde{s})$ binds occurrences of session channels in $\tilde{s}$ in α, which induces the standard alpha-equality.

A *typing judgement* has the following form:

$$\Gamma \vdash I \rhd \Delta$$

where Γ is *service typing* and Δ *session typing*. Δ maps session channels to their locations and session types and Γ located service channels and recursive variables to service types and session typing, respectively. The grammar of service/session typings follow.

Below in $\tilde{s}[A, B]$ we assume $A \neq B$.

$$\Gamma \ ::= \emptyset \mid \Gamma,\ ch@A : (\tilde{s})\alpha \mid \Gamma,\ x@A : \theta \mid \Gamma,\ X : \Delta$$
$$\Delta \ ::= \emptyset \mid \Delta,\ \tilde{s}[A, B] : \alpha \mid \Delta,\ \tilde{s} : \bot$$

In a service typing, three forms of assignments are used: $ch@A : (\tilde{s})\alpha$ says that a service channel ch is located at A and offers a service interface represented by a service type $(\tilde{s})\alpha$; $x@A : \theta$ says that a variable x located at A may store values of type θ; finally, $X : \Delta$ is for recursion i.e. when the interaction recurs to X, the behaviour will own a session typing Δ.

A session typing uses the primary form of assignment $\tilde{s}[A, B] : \alpha$ which says that a vector of session channels $\tilde{s}$, all belonging to a same session which is between A and B, has the session type α when seen from the viewpoint of A. We write Γ_1, Γ_2 (Δ_1, Δ_2) if there is no overlap between the free variables/names in Γ_1 (Δ_1) and Γ_2 (Δ_2). The notation $\mathsf{fsc}(\Delta)$ denotes the set of free service/session channels in Δ.

The typing rules are given in Table 2. Rule (G-TCom) states that I should contain a session type α_j between A and B such that its session channels contain s. The communicated value e is typed in the source (A) while the variable x is typed in the target (B), with the same type θ_j. In the conclusion, a branching type should include the operator op_j whose value type is θ_j. In (G-TCom), the session type in focus is given with the direction from A to B, i.e. it abstracts the structure of the interaction in this session from the viewpoint of A. While this is consistent, we may also regard it from the receiver viewpoint (B). Thus we have the symmetric variant (G-TComInv).

Rule (G-TPar) uses the linearity condition found in [18]. This is done by the operator $\bullet$ where $\tilde{s}[A, B] : \alpha \in \Delta_1 \bullet \Delta_2$ iff either

1. $\tilde{s}[A, B] : \alpha_1 \in \Delta_1$, $\{\tilde{s}\}[A, B] : \alpha_2 \in \Delta_2$ and $\alpha = \alpha_1 \mid \alpha_2$;
2. $\tilde{s}[A, B] : \alpha \in \Delta_1$ and $\{\tilde{s}\} \cap \mathsf{fsc}(\Delta_2) = \emptyset$, or its symmetric case;

Note different session channels can be used in parallel, while service channels can be shared by multiple threads of interactions.

Rule (G-TInit) types session initiation. Since $\tilde{s}$ is to be abstracted as session channels belonging to a single session, we demand that there is a $ch@B : (\tilde{s})\alpha$ in the typing environment. Since $\tilde{s}$ is directed from B to A, α designates a session type seen from B's viewpoint resulting in $s\,[B, A] : \alpha$ where both A and B need be mentioned since a session is always between two parties. Note that $ch@B : (\tilde{s})\alpha$ is also assumed in the premise since ch may have already been used elsewhere (as a service channel can be shared).

In (G-TRes1), hiding of session names is introduced after the session initiation so that they can no longer be abstracted by (G-TInit). α is no longer necessary, so we replace it with $\bot$. Rule (G-TRes2) is used for removing unnecessary hidden session names one by one: when $\tilde{s}$ is empty, we take it off with (G-TRes3).

In Rule (G-TZero) we demand each session type used in the conclusion is a distinct vector of session channels and Γ is well-formed. A service type Γ is *well-formed* whenever $ch@A_i : (\tilde{s}_i)\alpha_i \in \Gamma$ $(i = 1, 2)$ implies $A_1 = A_2$ and $(\tilde{s}_1)\alpha_1 = (\tilde{s}_2)\alpha_2$. Moreover, $x@A_i : \theta_i, X : \Delta_i \in \Gamma$ implies $\theta_1 = \theta_2$, $A_1 = A_2$ and $\Delta_1 = \Delta_2$. Similarly, a session typing Δ is *well-formed* when for all $\tilde{s}_1[A_1, B_1]\alpha_1$ and $\tilde{s}_2[A_2, B_2]\alpha_2$ in Δ such that $\{\tilde{s}_1\} \cap \{\tilde{s}_2\} \neq \emptyset$ we have $\tilde{s}_1 = \tilde{s}_2$, $A_1 = A_2$, $B_1 = B_2$ and $\alpha_1 = \alpha_2$.

Proposition 1. $\Gamma \vdash I \rhd \Delta$ *implies* Γ *and* Δ *are well-formed.*

As a simple example, we type the Buyer-Seller interaction I. Service channel QuoteCh is assigned with service type

$$\begin{array}{l}(s)\ s \blacktriangleleft \mathsf{Quote}(\texttt{integer}).\ (\\ \quad s \blacktriangleright \mathsf{QuoteAccept}(\texttt{null}).\ s \blacktriangleleft \mathsf{DeliveryDetails}(\texttt{null}).\ \mathsf{end} + \\ \quad s \blacktriangleright \mathsf{QuoteReject}(\texttt{string}).\ \mathsf{end})\end{array}$$

Instead, service channel DeliveryCh has service type

$$(t)\ t \blacktriangleleft \mathsf{DeliveryDetails}(\texttt{string}).\ \mathsf{end}$$

Denoting two types by $(s)\alpha_1$ and $(t)\alpha_2$ respectively, we can prove $\mathsf{QuoteCh} : (s)\alpha_1,\ \mathsf{DeliveryCh} : (t)\alpha_2 \vdash I \rhd \emptyset$.

The type discipline has also a minimal typing. To formulate minimality, we use the *inclusion ordering* $\leq$, defined based on simulation as in [16] with the key justifying rules being:

$$\frac{J \subset J' \qquad \forall i \in J.\ \alpha_i \leq \alpha'_i}{s \blacktriangleright \Sigma_{i\in J} op_i(\theta_i).\ \alpha_i \ \leq\ s \blacktriangleright \Sigma_{i\in J'} op_i(\theta_i).\ \alpha'_i}$$

$$\frac{J \subset J' \qquad \forall i \in J.\ \alpha_i \leq \alpha'_i}{s \blacktriangleleft \Sigma_{i\in J} op_i(\theta_i).\ \alpha_i \ \leq\ s \blacktriangleleft \Sigma_{i\in J'} op_i(\theta_i).\ \alpha'_i}$$

The relation $\leq$ is extended pointwise to session typings and service typings. In brief, $\alpha \leq \alpha'$ indicates α is the result of cutting off some branches from α' at zero or more points. We now observe:

Proposition 2. *Let* $\Gamma \vdash I \rhd \emptyset$ *for some* Γ. *Then there exists* Γ_0 *such that* $\Gamma_0 \vdash I$ *and whenever* $\Gamma' \vdash I \rhd \emptyset$ *we have* $\Gamma_0 \leq \Gamma'$.

Theorem 3 (Subject Reduction). *Assume* $\Gamma \vdash \sigma$. *Then* $\Gamma \vdash I \rhd \Delta$ *and* $(\sigma, I) \to (\sigma', I')$ *imply* $\Gamma \vdash \sigma'$ *and* $\Gamma \vdash I \rhd \Delta'$ *for some* Δ' *such that* $\mathsf{fsc}(\Delta') \subset \mathsf{fsc}(\Delta)$.

4 The End-Point Calculus

4.1 Syntax

The end-point calculus is an applied version of the π-calculus [25]. The main syntactic terms are *processes* ($P, Q, \ldots$) and *networks* ($M, N, \ldots$) and are defined by the following grammar.

$$\begin{array}{rcl} P & ::= & !\,ch(\tilde{s}).\ P \ \mid\ \overline{ch}(\nu\tilde{s}).\ P \ \mid\ s \rhd \Sigma_i \mathsf{op}_i(\tilde{y}_i).P_i \\ & & \mid \overline{s} \lhd \mathsf{op}\langle\tilde{e}\rangle.\ P \ \mid\ x := e.\ P \ \mid\ \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \\ & & \mid P \oplus P \ \mid\ P \mid Q \ \mid\ (\nu s)\ P \ \mid\ X \ \mid\ \mu X.\ P \ \mid\ \mathbf{0} \\ N & ::= & A[\ P\ ]_\sigma \ \mid\ N \mid N \ \mid\ (\nu s)\ N \ \mid\ \epsilon \end{array}$$

Table 3 Semantics of the End-Point Calculus

$$\text{(EP-Init)}\ \frac{-}{A[\,!\,ch(\tilde{s}).\,P \mid P'\,]_\sigma \mid B[\,\overline{ch}(\nu\tilde{s}).\,Q \mid Q'\,]_{\sigma'} \;\to\; (\nu\tilde{s})\,(A[\,!\,ch(\tilde{s}).\,P \mid P \mid P'\,]_\sigma \mid B[\,Q \mid Q'\,]_{\sigma'})}$$

$$\text{(EP-Com)}\ \frac{\sigma \vdash e \Downarrow v}{A[\,s \rhd \Sigma_i \mathsf{op}_i(x_i).P_i \mid P'\,]_\sigma \mid B[\,\overline{s} \lhd \mathsf{op}_j\langle e\rangle Q \mid Q'\,]_{\sigma'} \;\to\; A[\,P_j \mid P'\,]_{\sigma[x\mapsto v_j]} \mid B[\,Q \mid Q'\,]_{\sigma'}}$$

$$\text{(EP-Assign)}\ \frac{\sigma \vdash e \Downarrow v}{A[\,x := e.\,P \mid P'\,]_\sigma \;\to\; A[\,P \mid P'\,]_{\sigma[x\mapsto v]}}$$

The first two productions for processes describe terms meant for session initiation and the following two are for communication. This is in the style of [18], except $\tilde{y}_i$ in the second construct (branching input) do *not* induce binders. $x := e.\,P$ assigns a value v to x in its store and then continues as P. The rest is all standard. Networks are parallel composition of participants. The latter are represented by the term $A[\,P\,]_\sigma$ which indicates a participant A whose behaviour is given by P and whose local state is σ.

4.2 Reduction

The reduction semantics for the end-point calculus follows the π-calculus. We report the full definition in [1], but list the three key rules in Table 3.

(EP-Init) defines the session initiation: two participant A and B will synchronize starting a session whenever they are executing a service offer $!\,ch(\tilde{s}).\,P$ and a request of service $\overline{ch}(\nu\tilde{s}).\,Q$ respectively. The synchronisation will result into sharing fresh session names $\tilde{s}$ local to A and B. These session names are then used in (EP-Com) for communication. In (EP-Com), we use assignment to local variables rather than value passing for correspondence with the global calculus. (EP-Assign) updates the store associated in each participant.

4.3 Session Types for the End-Point Calculus

As mentioned above, the aim of the end-point calculus is to give a model on which we can project the global calculus. For this reason we need to define session types [18] as well. We use the same set of session and service types as the global calculus.

In the end-point calculus, we have the two type judgements

$$\Gamma \vdash_A P \rhd \Delta \qquad\qquad \Gamma \vdash M \rhd \Delta$$

respectively for processes and networks. Γ (service typing) and Δ (session typing) above are given by the following grammar.

$$\begin{aligned}\Gamma \;::=\; & \emptyset \mid \Gamma,\, ch@A : (\tilde{s})\alpha \mid \Gamma,\, \overline{ch}@A : (\tilde{s})\alpha \\ & \mid \Gamma,\, x@A : \theta \mid \Gamma,\, X : \Delta \\ \Delta \;::=\; & \emptyset \;\mid\; \Delta,\, \tilde{s}@A : \alpha \;\mid\; \Delta,\, \tilde{s} : \bot\end{aligned}$$

Table 4 Session Types for Processes in the End-Point Calculus

$$\text{(EP-TB)}\ \frac{K \subseteq J \quad s \in \tilde{s} \quad \Gamma \vdash x_j : \theta_j \quad \Gamma \vdash_A P_j \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash s \rhd \Sigma_{i\in J}\mathsf{op}_i(x_i).P_i \rhd \Delta \cdot \tilde{s}@A : s \blacktriangleright \Sigma_{i\in K}\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$\text{(EP-TS)}\ \frac{j \in J \subseteq K \quad \Gamma \vdash e : \theta_i \quad \Gamma \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \alpha_j}{\Gamma \vdash_A \overline{s} \lhd \mathsf{op}_j\langle e\rangle.P \rhd \Delta \cdot \tilde{s}@A : s \blacktriangleleft \Sigma_{i\in K}\mathsf{op}_i(\theta_i).\ \alpha_i}$$

$$\text{(EP-TSERV)}\ \frac{\Gamma \vdash_A P \rhd \tilde{s}@A : \alpha}{\Gamma,\ ch@A : (\tilde{s})\alpha \vdash_A \mathbf{!}\, ch(\tilde{s}).\ P \rhd \emptyset}$$

$$\text{(EP-TREQ)}\ \frac{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash_A P \rhd \Delta \cdot \tilde{s}@A : \alpha}{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash_A \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \rhd \Delta}$$

$$\text{(EP-TPAR)}\ \frac{\Gamma_i \vdash_A P_i \rhd \Delta_i \quad \Gamma_1 \asymp \Gamma_2 \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash_A P_1 \mid Q_2 \rhd \Delta_1 \odot \Delta_2}$$

As before, we stipulate that, whenever we write e.g. Γ_1, Γ_2, there are *no* free channels/variables shared between two typings. The selected rules for the typing processes are given in Table 4.

The rule (EP-TBRANCH) for input in-session communication involves branching with distinct operators: the typing can have less branches than the real process, so that the process is prepared to receive any operator specified in the type. Rule (EP-TSEL) is its dual: the typing can have more branches than the real process, so that the process invokes with operators at most those specified in types. Combining (EP-TB) and (EP-TS), an output never tries to invoke a non-existent option in its matching input.

Rule (EP-TSERV) is for typing the inputting side of initialisation. Note we do not allow those session channels other than the target of initialisation to be present as the session typing in the premise: this prevents *free* session channels to be under the replicated input, guaranteeing their linear usage. The typing in the conclusion means (by our convention) that ch or $\overline{ch}$ does not occur in Γ. The outputting side of initialisation (rule (EP-TREQ)) is analogous, except that the linearity constraint needs not be specified. We assume that A and B are not identical. The fact we allow $\overline{ch}@B : (\tilde{s})\alpha$ to occur in the premise means an invocation to a service can be done as many times as needed (as far as it is type correct). (EP-TPAR) uses the operators $\asymp$ and $\odot$: $\Delta_1 \asymp \Delta_2$ means two channels of the same domain have a dual type each other [18]; and the result of the composition of the dual types become $\bot$, which denotes the same channel cannot be composable further. This operation guarantees a linear use of session channels. The full definition can be found in [1].

As an example, we type the end-point process of the seller, seen in Section 2. If we consider the service types $(s)\alpha_1$ and $(t)\alpha_2$ in the previous example in Section 3.3, we have

$$\mathsf{QuoteCh} : (s)\alpha_1,\ \overline{\mathsf{DeliveryCh}} : (t)\overline{\alpha}_2 \ \vdash\ \mathsf{Seller}[\ \mathsf{Protocol}\ ]_\sigma \rhd \emptyset$$

Note that, in the end-point types the service channel DeliveryCh is overlined: this is because the channel is located at the shipper's. This is not the case in the global calculus as we only have a global view of channels. With well-formedness similar to the global calculus, we have:

Proposition 4. $\Gamma \vdash M \triangleright \Delta$ *implies* Γ *and* Δ *are well-formed.*

For the end-point calculus, we consider a subtyping relation on session types following [16]. [4] This relation plays a basic role in our subsequent technical development. The subtyping is written $\alpha \preceq \beta$. Intuitively, $\alpha_1 \preceq \alpha_2$ indicates that α_1 is more gentle, or dually α_2 is less constrained, in behaviour. The subtyping relation is given based on simulation following [16], whose key justifying rules are:

$$\frac{J \supset J' \quad \alpha_i \preceq \beta_i}{s \blacktriangleright \Sigma_{i\in J} op_i(\theta_i).\, \alpha_i \;\leq\; s \blacktriangleright \Sigma_{i\in J'} op_i(\theta_i).\, \beta_i}$$

which says that if the initial input offers more options, and if subsequent behaviours are more gentle, then it is more gentle.

$$\frac{J \subset J' \quad \alpha_i \preceq \beta_i}{s \blacktriangleleft \Sigma_{i\in J} op_i(\theta_i).\, \alpha_i \;\leq\; s \blacktriangleleft \Sigma_{i\in J'} op_i(\theta_i).\, \beta_i}$$

which says that if the initial output has less emissions and if subsequent behaviours are more gentle then it is more gentle. Note this relation is different from the inclusion ordering $\leq$ in §3.3.

The following result says that we can always find a representative typing for a given process, and, moreover, we can do so effectively. Such a type is minimum among all assignable typings w.r.t. the subtyping relation, so that we call it the *minimal typing* of a given term. Below and henceforth we write $\Gamma \vdash M$ for $\Gamma \vdash M \triangleright \emptyset$, similarly for $\Gamma \vdash_A P$.

Definition 5 (minimal typing). *Let* $\Gamma_0 \vdash M$. *We call* Γ_0 *the* minimal service typing of M *whenever for all* Γ *such that* $\Gamma \vdash I$ *we have* $\Gamma_0 \preceq \Gamma$, *where* $\preceq$ *is taken pointwise at each channels.*

Proposition 6 (existence of minimal typing). *For each typable* M, *its minimal service typing* Γ_0 *exists. Further such* Γ_0 *is algorithmically calculable from* M.

Theorem 7 (subject reduction). *If* $\Gamma \vdash N \triangleright \Delta$ *and* $N \to N'$ *then* $\Gamma \vdash N' \triangleright \Delta$.

Unlike the global calculus, the untyped end-point calculus can have communication error. Its absence is guaranteed by the type system. Let us say M has a *communication error* when:

$$M \equiv C[A[s \triangleright \Sigma_i \mathsf{op}_i(x_i).P_i | R]_\sigma | B[\overline{s} \lhd \mathsf{op}\langle e\rangle.Q | S]_{\sigma'}]$$

where in both cases $\mathsf{op} \notin \{\mathsf{op}_i\}$ and $C[\ ]$ is a reduction context (i.e. a context whose hole is not under a prefix). That is, M has a communication error when it contains an input and an output at a common channel which however do not match in operator names. A basic corollary of Theorem 7 follows.

Corollary 8 (lack of communication error). *If* $\Gamma \vdash N \triangleright \Delta$ *and* $N \to^* M$, *then* M *never contains a communication error.*

[4] The direction of the subtyping is converse to (and consistent with) [16].

5 The End-Point Projection

This section establishes a formal link from the global calculus to the end-point calculus: a global description which conforms to the three properties, *connectedness, well-threadedness and coherency* can be mapped to the end-points preserving the three desirable properties, *type preservation*, and *soundness* and *completeness* of the operational semantics. Throughout we only consider well-typed terms for both the global and end-point calculi.

5.1 Connectedness

To define connectedness, we need to say which participant initiates an action in a given interaction I: this participant should be the place where the preceding event happens. First assume we annotate recursion variable with a participant name, e.g. $\mu X^A\mathbf{.}\ I$ etc.

Definition 9 (initiating participants). *Given a hiding-free interaction I, its* initiating participants*, denoted* $\mathsf{top}(I)$*, is inductively given as follows:*

$$\mathsf{top}(I) \stackrel{def}{=} \begin{cases} \{A\} & \text{if } I \in Z \\ \emptyset & \text{if } I \equiv \mathbf{0} \\ \mathsf{top}(I') & \text{if } I \equiv \mu X^A\mathbf{.}\ I' \\ \mathsf{top}(I_1) \cup \mathsf{top}(I_2) & \text{if } I \equiv I_1 \mid I_2 \text{ or } I_1 + I_2 \end{cases}$$

where $Z = \{\mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2, A \to B : ch(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I',\ A \to B : s\langle \mathsf{op},\ e,\ x\rangle\mathbf{.}\ I,\ x@A := e\mathbf{.}\ I',\ X^A\}$ *. If* $A \in \mathsf{top}(I)$*, we say A is an* initiating participant of I*.*

The map $\mathsf{top}(I)$ generates a set of participants that initiates the first action of I. We can now present the definition of connectedness.

Definition 10 (connectedness). The collection of *connected interactions* Con is inductively generated as follows.

1. $\{\mathbf{0},\ X^A\} \subseteq \mathsf{Con}$.
2. $A \to B : ch(\boldsymbol{\nu}\, s)\mathbf{.}\ I'$, $A \to B : s\langle \mathsf{op},\ e,\ x\rangle\mathbf{.}\ I'$, $\mu X^B\mathbf{.}\ I'$ and $x@B := e\mathbf{.}\ I'$ are in Con if $I' \in \mathsf{Con}$ and $\mathsf{top}(I') = \{B\}$.
3. $\mathsf{if}\ e@A\ \mathsf{then}\ I_1\ \mathsf{else}\ I_2$, $I_1 + I_2$ and $I_1 \mid I_2$ are in Con if $I_1,\ I_2 \in \mathsf{Con}$ and $\{A\} = \mathsf{top}(I_1) = \mathsf{top}(I_2)$.

Connectedness says that, in communication actions, only the message reception leads to activity (at the receiving participant), and that such activity should immediately follow the reception of messages. We note connectedness enjoys a subject reduction property, shared by well-threadedness and coherence.

5.2 Well-Threadedness

In order to formally introduce the notion of well-threadedness, we need to annotate a global interaction with threads.

Definition 11 (annotated interaction). *Annotated interactions*, denoted by $\mathcal{A}, \mathcal{A}', \ldots$, are given by the following grammar.

$$\begin{aligned}\mathcal{A} ::= \;& A^{\tau_1} \to B^{\tau_2} : ch(\boldsymbol{\nu}\, \tilde{s}).\, \mathcal{A} \mid x@A^{\tau} := e.\, \mathcal{A} \;\mid\; \mathcal{A}_1 \,|^{\tau}\, \mathcal{A}_2 \\ & \mid A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op},\ e,\ y\rangle.\, \mathcal{A} \mid \mu^{\tau} X^{A}.\, \mathcal{A} \;\mid\; \mathcal{A}_1 +^{\tau} \mathcal{A}_2 \\ & \mid \mathsf{if}\ e@A^{\tau}\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2 \mid X^{A}_{\tau} \mid \mathbf{0}\end{aligned}$$

where $\tau_i \in \mathbb{N}$ (called *thread*) and $\tau_1 \neq \tau_2$ in the first two lines.

In the abstract syntax tree of the terms in the global calculus, each node is annotated with threads (given as natural numbers). The notions such as connectedness easily extend to annotated interactions. The following is an annotated interaction of a previous example:

$$\begin{aligned}&\mathsf{Buyer}^1 \to \mathsf{Seller}^2 : ch_1(\boldsymbol{\nu}\, s).\ \mathsf{Seller}^2 \to \mathsf{Buyer}^3 : ch_2(\boldsymbol{\nu}\, t).\\ &\quad \mathsf{Buyer}^3 \to \mathsf{Seller}^2 : t\langle \mathsf{op}_1,\ v_1,\ x\rangle.\\ &\qquad \mathsf{Seller}^2 \to \mathsf{Buyer}^1 : s\langle \mathsf{op}_2,\ v_2,\ y\rangle\end{aligned}$$

Our task now is to find a notion of "consistent annotation" for annotated interactions, so that causality specified globally can be precisely realisable locally. For this purpose it is convenient to consider each $\mathcal{A}$ as an inverted abstract syntax tree. Each node has a *constructor* which is annotated by either one thread or, if it is initiation or communication, an ordered pair of threads.

Definition 12. 1. If a node in $\mathcal{A}$ is initialisation or communication from B to C and is annotated by (τ_1, τ_2), then τ_1 (resp. τ_2) is the *active (resp. passive) thread by B (resp. by C)* of that node. If the node has other constructors, its annotating thread is both active and passive.
2. If a node occurs (resp. directly) above some node, then the former is a *(resp. direct) predecessor* of the latter. Symmetrically we define *(direct) successor*.

Note if a node is a predecessor of another then the former execution should temporarily precedes that of the latter. We can now introduce the consistency condition for thread annotation. Below in (G2) we assume the bound name condition for session names.

Definition 13. An annotated connected interaction $\mathcal{A}$ is *globally consistent* or simply *consistent* if the following conditions hold.

(G1) Freshness Condition: For each node of $\mathcal{A}$, if it starts with initialisation, then its passive thread should be fresh w.r.t. all of its predecessors (if any).
(G2) Session Consistency: If a node of $\mathcal{A}$ starts with a communication between B and C via (say) s and another node of $\mathcal{A}$ starts with a communication via s or an initialisation which opens s, then the thread by B (resp. by C) of the former node should coincide with the thread by B (resp. by C) of the latter node.
(G3) Causal Consistency: If a node of $\mathcal{A}$ is the direct successor of another node of $\mathcal{A}$, then the latter's active thread should coincide with the former passive thread.

(G1) says a fresh thread starts when a service is invoked. **(G2)** says two distinct interactions in the same session (which are, by typing, always between the same pair of participants) should be given the same threads w.r.t. each participant. **(G3)** says if A has an input annotated as a (passive) thread then its immediately following output should be annotated by the same (but this time active) thread.

Below we say *I has an annotation $\mathcal{A}$* when removing all annotations from $\mathcal{A}$ coincides with I.

Definition 14 (well-threaded interactions). I is *well-threaded* when it is connected and has a consistent annotation.

Note well-threadedness implies connectedness (hence well-typedness). In [1], we give a type discipline accepting all and only well-threaded interactions, from which we can derive a sound and complete inductive algorithm to check well-threadedness.

5.3 Coherence and End-Point Projection

We now define coherence and then end-point projection. First, we give the notion of mergeability of threads. In the rest of the paper, a *typed term* (in the end-point calculus) is a typed sequent $\Gamma \vdash_A P \triangleright \Delta$ or $\Gamma \vdash M \triangleright \Delta$. Moreover, a relation over typed processes or networks (in the end-point calculus) is *typed* if each related pair of typed terms have the same typing.

Definition 15 (mergeability). *Mergeability relation*, denoted $\bowtie$, is the smallest typed equivalence over terms up to $\equiv$ closed under all typed contexts and the rule:

$$\frac{\forall i \in J \cap K.\ P_i \bowtie Q_i \quad \forall j \in J \setminus K, k \in K \setminus J.\ op_j \neq op_k}{s \triangleright \Sigma_J op_j(x_j).\ P_j \ \bowtie \ s \triangleright \Sigma_K op_k(x_k).\ Q_k}$$

When $P \bowtie Q$, we say *P and Q are mergeable*.

The relation $\bowtie$ checks whether two given processes behave without contradicting when they come to the same course of interactions, i.e. when the same input branch is selected by the interacting party. Thus the rules above say that we can allow differences in input branches which do not overlap, but we do demand each pair of behaviours with the same operation to be identical.

Definition 16 (merge operator). $\sqcup$ is a partial commutative binary operator on processes, such that (1) if P is a prefixed process with a service channel as its subject, then $P \sqcup \mathbf{0} = \mathbf{0} \sqcup P = P$; and (2) $s \triangleright \Sigma_{i \in J} \mathsf{op}_i(y_i).\ P_i \sqcup s \triangleright \Sigma_{i \in K} \mathsf{op}_i(y_i).\ Q_i$ is defined as:

$$\begin{array}{l} \Sigma_{i \in J \cap K} \mathsf{op}_i(y_i).\ (P_i \sqcup Q_i) \ + \Sigma_{i \in J \setminus K} \mathsf{op}_i(y_i).\ P_i \ + \\ \quad + \Sigma_{i \in K \setminus J} \mathsf{op}_i(y_i).\ Q_i \end{array}$$

where we assume that every time $\sqcup$ is applied in the defining clause, say to P and Q, we have $P \bowtie Q$; and otherwise it is the identity. When these conditions are not satisfied, the operation is undefined.

Before introducting projection onto threads, we add a further annotation to each recursion and each recursion variable. Given $\mu^{\tau} X^{A}.\mathcal{A}$ in an annotated interaction, let $\{\tau_i\}$ be the set of threads occurring in, but *not* initiated in, $\mathcal{A}$. Then we further annotate this recursion as $\mu^{\tau:\{\tau_i\}}X$ and each free X^{A}_{τ} in $\mathcal{A}$ as $X^{A}_{\tau:\{tau_i\}}$. The added information is used for taking off unnecessary recursion from endpoint processes.[5]

Definition 17 (thread projection). Let $\mathcal{A}$ be consistently annotated. Then the partial operation $\mathsf{TP}(\mathcal{A}, \tau)$ is defined as follows:

- $\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : b(\nu\, \tilde{s})\mathbf{.}\, \mathcal{A}, \tau) \stackrel{\text{def}}{=}$
$$\begin{cases} \overline{b}(\nu \tilde{s})\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau_1) & \text{if } \tau = \tau_1 \\ \mathbf{!}\, b(\tilde{s})\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau_2) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$
- $\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op_i},\ e_i,\ x_i\rangle\mathbf{.}\, \mathcal{A}, \tau) \stackrel{\text{def}}{=}$
$$\begin{cases} \overline{s} \lhd \mathsf{op}\langle e\rangle\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_1 \\ s \rhd \mathsf{op_i}(x_i)\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$
- $\mathsf{TP}(\mathsf{if}\ e@A^{\tau'}\ \mathsf{then}\ \mathcal{A}_1\ \mathsf{else}\ \mathcal{A}_2, \tau) \stackrel{\text{def}}{=}$
$$\begin{cases} \mathsf{if}\ e\ \mathsf{then}\ \mathsf{TP}(\mathcal{A}_1, \tau')\ \mathsf{else}\ \mathsf{TP}(\mathcal{A}_2, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1, \tau) \sqcup \mathsf{TP}(\mathcal{A}_2, \tau) & \text{otherwise} \end{cases}$$
- $\mathsf{TP}(x@A^{\tau'} := e\mathbf{.}\, \mathcal{A}, \tau) \stackrel{\text{def}}{=}$
$$\begin{cases} x := e\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$
- $\mathsf{TP}(\mathcal{A}_1 \star^{\tau'} \mathcal{A}_2, \tau) \stackrel{\text{def}}{=}$
$$\begin{cases} \mathsf{TP}(\mathcal{A}_1, \tau') \star \mathsf{TP}(\mathcal{A}_2, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1, \tau) \sqcup \mathsf{TP}(\mathcal{A}_2, \tau) & \text{otherwise} \end{cases}$$
- $\mathsf{TP}(\mu^{\tau':\{\tilde{\tau}_i\}}X^{A}\mathbf{.}\, \mathcal{A}, \tau) \stackrel{\text{def}}{=} \mu X\mathbf{.}\, \mathsf{TP}(\mathcal{A}, \tau)$ if $\tau \in \{\tilde{\tau}_i\}$, $\mathbf{0}$ if else.
- $\mathsf{TP}(X^{A}_{\tau:\{\tilde{\tau}_i\}}, \tau) \stackrel{\text{def}}{=} X$ if $\tau \in \{\tilde{\tau}_i\}$, $\mathbf{0}$ if else.
- $\mathsf{TP}(\mathbf{0}, \tau) \stackrel{\text{def}}{=} \mathbf{0}$

where $\star \in \{\oplus, |\}$. If $\mathsf{TP}(\mathcal{A}, \tau)$ is undefined then we set $\mathsf{TP}(\mathcal{A}, \tau) = \bot$.

Note the thread projection already uses the definedness of the $\sqcup$ operator. The notion of coherence assumes this thread-level mergeability, extending it to inter-thread consistency. As noted in §2, the need for inter-thread consistency arises because the description of the behaviour of a service may as well be scattered over more than one places in a global description. Since each service channel *ch* uniquely defines a service, we can collect all threads contributing to its behaviour by taking the passive thread of each session initialisation at *ch*.

We now define coherence of well-threaded annotated interaction. Below, given an annotated interaction $\mathcal{A}$, we write $\tau_1 \equiv_{\mathcal{A}} \tau_2$ whenever τ_1 and τ_2 in $\mathcal{A}$ belong to the same service channel.

[5] The use of this added information does not affect behaviour, but is needed for type preservation. The added annotation is only used for thread projection.

Definition 18 (coherence). We say $\mathcal{A}$ is *coherent* if it is consistently annotated (hence well-threaded) and satisfies:

1. For each thread τ in $\mathcal{A}$, $\mathsf{TP}(\mathcal{A}, \tau)$ is well-defined.
2. For each pair of threads τ_1, τ_2 in $\mathcal{A}$ with $\tau_1 \equiv_{\mathcal{A}} \tau_2$, we have $\mathsf{TP}(\mathcal{A}, \tau_1) \bowtie \mathsf{TP}(\mathcal{A}, \tau_2)$.

Proposition 19. *Given a well-typed I, it is decidable whether I is coherent (hence connected and well-threaded) or not.*

We can now define the endpoint projection.Below we call an interaction *I restriction-free* whenever it contains no terms of the form $(\boldsymbol{\nu} s)\ I'$ as its subterm.

Definition 20 (end-point projection). Let I be a restriction-free and coherent interaction with free session names $\tilde{s}$ and let $\mathcal{A}$ be one of its consistent annotations. Then the end point projection of $(\boldsymbol{\nu}\tilde{s})\ I$ under σ, denoted $\mathsf{EPP}((\boldsymbol{\nu}\tilde{s})\ I, \sigma)$, is given as the following network.

$$(\boldsymbol{\nu}\tilde{s})\ \Pi_{A \in \mathsf{part}(I)}\ A[\![\ \Pi_{[\tau]} \bigsqcup_{\tau' \in [\tau]} \mathsf{TP}(\mathcal{A}, \tau')\ ]\!]_{\sigma@A}$$

where $\mathsf{part}(I)$ denotes the set of participants mentioned in $\mathcal{I}$.

5.4 Pruning and Main Theorems

Suppose we have an interaction composed by two branches where the first two interactions are $\mathsf{Buyer} \to \mathsf{Seller} : ch(\boldsymbol{\nu}\, s)$**.** $\mathsf{Seller} \to \mathsf{Buyer} : s\langle\mathsf{ack}\rangle$ and then in one branch we have $\mathsf{Buyer} \to \mathsf{Seller} : s\langle\mathsf{go}\rangle$ and in the other $\mathsf{Buyer} \to \mathsf{Seller} : s\langle\mathsf{stop}\rangle$. We then get that Buyer and Seller are respectively projected as

$$\overline{ch}(\boldsymbol{\nu} s).s \rhd \mathsf{ack}().\overline{s} \lhd \mathsf{go}\langle\rangle \oplus \overline{ch}(\boldsymbol{\nu} s).s \rhd \mathsf{ack}().\overline{s} \lhd \mathsf{stop}\langle\rangle$$
$$\mathbf{!}\, ch(s).\overline{s} \lhd \mathsf{ack}\langle\rangle.(s \rhd \mathsf{ok}() \ +\ s \rhd \mathsf{stop}())$$

By the dynamics of the choice operator, dropping one branch reduces to $\mathsf{Seller} \to \mathsf{Buyer} : s\langle\mathsf{ack}\rangle$**.** $\mathsf{Buyer} \to \mathsf{Seller} : s\langle\mathsf{go}\rangle$. Its end-point projection is the network:

$$\begin{array}{l} \mathsf{Buyer}[\![\ \overline{ch}(\boldsymbol{\nu} s).s \rhd \mathsf{ack}().\overline{s} \lhd \mathsf{go}\langle\rangle\ ]\!]_{\sigma@\mathsf{Buyer}} \quad \mid \\ \mathsf{Seller}[\![\ \mathbf{!}\, ch(s).\overline{s} \lhd \mathsf{ack}\langle\rangle.s \rhd \mathsf{go}()\ ]\!]_{\sigma@\mathsf{Seller}} \end{array} \tag{2}$$

However the original end-point projection reduces as:

$$\begin{array}{l} \mathsf{Buyer}[\![\ \overline{ch}(\boldsymbol{\nu} s).s \rhd \mathsf{ack}().\overline{s} \lhd \mathsf{go}\langle\rangle\ ]\!]_{\sigma@\mathsf{Buyer}} \quad \mid \\ \mathsf{Seller}[\![\ \mathbf{!}\, ch(s).\overline{s} \lhd \mathsf{ack}\langle\rangle.(s \rhd \mathsf{go}() \oplus s \rhd \mathsf{stop}())\ ]\!]_{\sigma@\mathsf{Seller}} \end{array} \tag{3}$$

There is discrepancy between (2) and (3): the former has *lost* one branch, while (3) keeps it. Notice this lost branch is inessential from the viewpoint of the internal dynamics of the configuration: "stop" is never used the global description obtained from reduction.

This example shows that a global interaction can lose information during reduction which is still kept in the corresponding reduction in its EPP, due to persistent behaviour at service channels. This motivates the introduction of the asymmetric relation of *pruning* that we shall use to state a property of the end-point projection. Below we write $\mathbf{!}R$ when R is a n-fold composition of replications.

Definition 21 (pruning). Assume $\Gamma \vdash_A P \triangleright \Delta$, $\Gamma, \Gamma' \vdash_A Q \triangleright \Delta$ and, moreover, $\Gamma \vdash_A P \triangleright \Delta$ is a minimal typing. If further we have $Q \equiv Q_0 | !R$ where $\Gamma \vdash Q_0 \triangleright \Delta$, $\Gamma' \vdash_A R$ and $P \bowtie Q_0$, then we write $\Gamma \vdash_A P \prec Q \triangleright \Delta$ or $P \prec Q$ for short; and say P *prunes* Q.

The pruning $P \prec Q$ indicates P is the result of cutting off "unnecessary branches" of Q, in the light of P's own typing. $\prec$ is in fact a typed strong bisimulation in the sense that $P \prec Q$ means they have precisely the same observable behaviours *except for the visible input actions at pruned inputs, either branches or replicated channels.* Thus in particular it satisfies the following condition.

Lemma 22 (pruning lemma). *$\prec$ is a strong reduction bisimulation in the sense that it satisfies the following two clauses:*

1. *If $M \prec N$ and $M \to M'$ then $N \to N'$ such that $M' \prec N'$.*
2. *If $M \prec N$ and $N \to N'$ then $M \to M'$ such that $M' \prec N'$.*

Further $\prec$ is transitive.

As noted $\prec$ satisfies a stronger property of being a strong bisimulation w.r.t. typed transitions under the minimal typing of the l.h.s. processes. We have finally arrived at the main results of this paper.

Theorem 23 (End-Point Projection). *Assume I is coherent. Assume further $\Gamma \vdash I \triangleright \Delta$ and $\Gamma \vdash \sigma$. Then we have:*

1. (type preservation) *If $\Gamma \vdash I \triangleright \Delta$ is the minimal typing of I, then $\Gamma \vdash \mathsf{EPP}(I, \sigma) \triangleright \Delta'$ where Δ' is the result of replacing each occurrence of type assignment in Δ, say $\tilde{s}@A : \alpha$, with $\tilde{s} : \bot$. In particular, if $\Gamma \vdash I$ then $\Gamma \vdash \mathsf{EPP}(I, \sigma)$.*
2. (soundness) *if $\mathsf{EPP}(I, \sigma) \to N$ then there exists I' such that $(\sigma,\ I) \to (\sigma',\ I')$ and $\mathsf{EPP}(I', \sigma') \prec N$.*
3. (completeness) *If $(\sigma,\ I) \to (\sigma',\ I')$ then $\mathsf{EPP}(I, \sigma) \to N$ such that $\mathsf{EPP}(I', \sigma') \prec N$.*

Proof Outline. For (1), type preservation, we use an auxiliary typing system for annotated interactions that infers a minimal typing for each thread. This per-thread minimal typing coincides with the minimal typing of the projection of the corresponding thread in the endpoint calculus. Further the minimal typing of the whole global term is the result of merging all of its per-thread minimal typings, similarly for the endpoint calculus.

For (2) and (3), consider coherent $I = \Pi_i I_i$ and its projection:

$$M \stackrel{\text{def}}{=} A[\{P_l\}_{l \in L}]_{\sigma_A} \mid B[\{Q_m\}_{m \in M}]_{\sigma_B} \mid C[\{R_n\}_{n \in N}]_{\sigma_C}$$

Above for simplicity we consider only three participants, ignore hiding, and let each P_l etc. be a thread projection (the reasoning is similar in the general case). For (2), soundness, assume $M \to M'$. By induction there is the corresponding redex in I. The rest is case analysis of the redex, taking the results of reducing I_i of I and either a thread (if it is not interaction) or a pair of threads (if it is). We then collect all threads again and compare the results. As a simple case, $P_l \stackrel{\text{def}}{=} x := e.P'_l$ results in P'_l with an altered

state, while $I_i \stackrel{\text{def}}{=} x@A := e.I_i'$ results in I_i' with the same state. The projection of I_i' is the same as the projection of I_i except it loses $x := e$ from the corresponding thread, in this case P_l, that is we get P_l'. For all other cases it is possible the projection can lose some branches or the whole replicated process, which we equate by $\prec$. (3), completeness, is by a similar reasoning. For details, see [13]. □

By Corollary 8 and lemma 22, Theorem 23 immediately implies:

Corollary 24.

1. (error freedom) *If* $\Gamma \vdash I$ *and* $\Gamma \vdash \sigma$, *then* $\mathsf{EPP}(I, \sigma)$ *has no communication error.*
2. (soundness) *if* $\mathsf{EPP}(I, \sigma) \rightarrow^n N$ *then there exists* I' *such that* $(\sigma,\ I) \rightarrow^n (\sigma',\ I')$ *and* $\mathsf{EPP}(I', \sigma') \prec N$.
3. (completeness) *If* $(\sigma,\ I) \rightarrow^n (\sigma',\ I')$ *then* $\mathsf{EPP}(I, \sigma) \rightarrow^n N$ *such that* $\mathsf{EPP}(I', \sigma') \prec N$.

6 Extensions and Applications of EPP Theory

6.1 Local variable declaration.

We consider extensions and applications of the theory of EPP. First, we augment the syntax of global/local calculi with one useful construct, *local variable declaration*:

$$\texttt{newvar}\ x@A := e\ \texttt{in}\ I \qquad \texttt{newvar}\ x := e\ \texttt{in}\ P$$

This construct is indispensable especially for repeatedly invocable behaviours, i.e. those of services. Suppose a bookseller is invoked by two buyers simultaneously, each asking a quote for a different book. If these two threads share a variable, these two requests will get confused. The use of local variable declaration can avoid such confusion. The dynamics and typing of this construct are standard [29]. For endpoint projection, it is treated just as assignment.

6.2 Intra-Participant Interaction.

In §3.3, we demanded that, in the grammar of service typing, $A \neq B$ in $\tilde{s}[A, B]$. This means well-typed global terms never have an intra-participant interaction. This is a natural assumption in a business protocol which primarily specifies inter-organisational interactions: however it can be restrictive in other contexts. Under connectedness (whose definition does not change), we can easily adapt the EPP theory to the inclusion of intra-participant interactions. First, the typing rules in Table 2, page 13, takes off (G-TComInv) and refines (G-TCom) so that the typing $\tilde{s}[A, B] : \alpha$ always reflects the direction of the interaction just inferred. This allows us to treat the case when A and B are equal. The key change is in well-threadedness. When $A = B$, the condition (G2) (session consistency) in Definition 13 is problematic since we do not know which of the two threads should be given to which participant. However stipulating the following condition solves this ambiguity:

Local Causal Consistency: If there is a downward sequence of actions which starts from an active thread τ and ends with an action in which τ occurs for the first time (i.e. τ occurs in no intermediate actions in the sequence), then the latter τ occurs passively.

We also note this condition is a *consequence* of (G1–3) in the theory without intra-participant interaction so that we are not adding any extra constraint to inter-participant interactions.

6.3 Name Passing.

An extension which is technically significant and practically useful is the introduction of *channel passing*. Channel passing is often essential in business protocols. As an example, consider the following refinement of Buyer-Seller Protocol.

> Buyer *wants to buy a hardware from* Seller*, but* Buyer *knows no* Seller*'s address on the net, i.e. it does not know* Seller*'s service channel. The only thing* Buyer *knows is a service channel* hardware *of a* DirectoryService*, which will send back the address of a* Seller *to* Buyer *which in turn interacts with that* Seller *through the obtained channel.*

In such a situation, Buyer has no prior knowledge of not only the seller's channel but also the participant itself. In a global description including its typing, participant names play a basic role. Can we leave the name of a participant and its channels unknown and still have a consistent EPP theory? This has been an open problem left in WS-CDL's current specification (which allows channel passing only for a fixed participant). Below we restrict our attention to service channel passing, excluding session name passing (which poses an additional technical issue [18]).

First, at the level of he endpoint calculus, it suffices to use the channel passing in the standard π-calculus.

$$\overline{\mathsf{DirectoryService}}(s).s(y).\overline{y}(t).P$$

which describes the initial behaviour of Buyer. Note y is an imperative variable, so that $y(t).P$ first *reads* the content of y then uses it for communication. The typing rules are extended accordingly.

In the global calculus, we introduce a syntactic variable Y, called a *participant placeholder*, for denoting anonymous participants. For example we can write:

$$A \to Y : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I \qquad\qquad Y \to Y' : s\langle \mathsf{op},\ e,\ y\rangle\mathbf{.}\ I$$

The newly added $A \to Y : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I$ intuitively says:

> *A starts a session with session names $\tilde{s}$ on the service channel stored in x at the location A.*

The participant at which the service is offered is left unknown by placing a placeholder Y. However this will be instantiated once the variable x at A is inspected. For example, if x is evaluated to $ch@B$ in the store, the interaction takes place as in $A \to B : ch(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I$.

As an example, we present the buyer-seller-directory scenario discussed above:

$$\begin{array}{l}
\mathsf{Buyer} \to \mathsf{Directory} : \mathsf{hardware}(\boldsymbol{\nu}\, s)\mathbf{.} \\
\quad \mathsf{Directory} \to \mathsf{Buyer} : s\langle \mathsf{sell},\ \mathsf{hware@amazon.co.uk},\ x\rangle\mathbf{.} \\
\quad (\ \mathsf{Buyer} \to Y : x(\boldsymbol{\nu}\, s')\mathbf{.}\ Y \to \mathsf{Buyer} : s'\langle \mathsf{OK},\ \mathsf{data},\ y\rangle\ \mid \\
\quad\ \ \mathsf{Buyer} \to \mathsf{Directory} : s\langle \mathsf{more},\ \text{""},\ z\rangle\mathbf{.} \\
\quad\ \ \mathsf{Directory} \to \mathsf{Buyer} : s\langle \mathsf{sell},\ \mathsf{hardware@pcworld.co.uk},\ x\rangle\mathbf{.} \\
\quad\ \ \mathsf{Buyer} \to Y' : x(\boldsymbol{\nu}\, s'')\mathbf{.}\ Y' \to \mathsf{Buyer} : s''\langle \mathsf{OK},\ \mathsf{data},\ y\rangle)
\end{array}$$

Note that, depending on the channel sent from Directory, Y and Y' are assigned to different participants.

The dynamics of the global calculus adds the rule which infers:

$$(\sigma,\ A \to Y : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I) \to (\sigma, (\boldsymbol{\nu} \tilde{s})\ I[B/Y])$$

whenever we have $\sigma@A(x) = ch@B$.

For types, we first extend the basic types θ with $(\tilde{s})\alpha$. We then add, with the obvious extension to the syntax of types:

$$\frac{\Gamma \vdash x@W_1 : (\tilde{s})\alpha \qquad \Gamma \vdash I \rhd \Delta \cdot \tilde{s}\,[W_2, W_1] : \alpha}{\Gamma \vdash W_1 \to W_2 : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I \rhd \Delta}$$

Other typing rules can be extended to deal with terms containing the participant variable Y in the same manner.

Finally, for the EPP theory, we need no change in the notion of connectedness. For well-threadedness, we first annotate placeholders regarding, e.g. $A \to Y : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I$ as the start of a new thread for Y, so we annotate it as $A^{\tau_1} \to Y^{\tau_2} : x(\boldsymbol{\nu}\, \tilde{s})\mathbf{.}\ I$ with τ_2 fresh. The definition of well-threadedness remains the same. Coherence however needs additional consideration. The variable $x@A$ can store different channels from different participants. For this purpose we use a typing system which records a possible set of assignment, in the shape $x@W_1 : C$ where C is a set of channels which may be instantiated into C. If some concrete channel is in C, the behaviour of that channel becomes constrained by coherence. This set C is inferred, starting from some fixed set, by adding ch (as in $x@W_1 : C \cup \{ch@B\}$) when we infer, e.g. $W_1 \to W_2 : s\langle \mathsf{op}_\mathsf{j},\ ch@B,\ x\rangle\mathbf{.}\ I$, where W_i can be either of participants or placeholders.

Leaving the technical details to [1, 13], we give a flavour of how this extension works by the end-point projection of the example above. We first consider the annotated interaction for placeholders.

$$\mathsf{Buyer}^1 \to Y^3 : x(\boldsymbol{\nu}\, s')\mathbf{.}\ Y^3 \to \mathsf{Buyer}^1 : s'\langle \mathsf{OK},\ \mathsf{data},\ y\rangle$$

In the projection of this thread, we have placed a hole $-$ which should be substituted with the appropriate service channels.

$$\mathsf{TP}(\mathcal{A}, 3) = \mathbf{!}\_(s')\mathbf{.}\ \overline{s'} \lhd \mathsf{OK}\langle \mathsf{data}\rangle$$

Thus, checking coherence consists in updating the definition of the function threads which induces the thread equivalence classes. But what equivalence classes should

threads 3 and 4 belong to? We can use the prediction of all the possible values x can assume at runtime, i.e. hware@amazon.co.uk and hardware@pcworld.co.uk. We have to make sure that thread 3 belongs to both threads($\mathcal{A}$, hware) and threads($\mathcal{A}$, hardware). Then, if we are end-point projecting in amazon.co.uk we will substitute hware to _ in both thread projections, and if we are end-point projecting pcworld.co.uk we will substitute hardware instead.

6.4 Conformance.

By relating global descriptions to their local counterpart, the presented theory allows us to make the best of the rich results from the study of process calculi. One such application is *conformance checking* (and its dynamic variant, runtime monitoring), discussed in Introduction. Our purpose is to have a formal criteria to say the communication behvaiour of a program P conforms to a global specification I.

Conformance concerns the possibility of checking whether an existing system tallies with a given specification. In process algebra and concurrency in general, this way of reasoning usually leads to system relations such as (inverse of) simulation or bisimulation. Given an implemented system, say P, the idea is to check whether P conforms to a well-typed specification in the global calculus. Then, using the end-point projection, we can generate an end-point network (which is in the same language as the given implemented system). This suggests that we must perform our comparison in the end-point calculus.

One interesting mechanism to be exploited is the typing system: the end-point projection generates not only a network consistent with the global specification, but also a type for the generated network. This can already be used for a first comparison with the given system: if this does not type, then the given system does not conform to the specification.

Unfortunately, there are cases where types may reveal as conform, systems which are not. Our solution is to adopt a notion of typed simulation, guaranteeing safety (for a stronger notion of conformance, see e.g. [11, 15]). Thus, the given system must be simulated by the specification with its minimal type in order conform to it.

Let us clarify this with an example in the buyer-seller scenario. Let P be the process

$$\overline{\mathsf{QuoteCh}}(\nu s).\ s \rhd \mathsf{Quote}(x).\\ \quad \mathsf{if}\ (x \leq 100)\ \mathsf{then}\ \overline{s} \lhd \mathsf{Accept}\langle\rangle\ \mathsf{else}\ \overline{s} \lhd \mathsf{Reject}\langle\rangle$$

Consider now a system (already implemented) with the following end-point processes (referred to as System):

$$\mathsf{Buyer}[P]\ \mid\ \mathsf{Seller}[\ !\,\mathsf{QuoteCh}(s).\ \overline{s} \lhd \mathsf{Quote}\langle 300\rangle.\\ s \rhd (\mathsf{Accept}()\ +\ \mathsf{Reject}()\ +\ \mathsf{Restart}())]$$

Suppose we want to check that the system above *conforms* to a specification given in the global calculus. The following specification says that the buyer either accepts or

rejects the quote.

$$\begin{array}{l}\mathsf{Buyer} \to \mathsf{Seller} : \mathsf{QuoteCh}(\nu\, s)\mathbf{.}\\ \quad \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{Quote},\ 300,\ x\rangle\mathbf{.}\\ \quad\quad \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{Accept}\rangle + \mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{Reject}\rangle\end{array}$$

We recall the end point projection of the specification above (referred to as Spec)::

$$\begin{array}{l}\mathsf{Buyer}[\ \overline{\mathsf{QuoteCh}}(\nu s)\mathbf{.}\ s \rhd \mathsf{Quote}(x)\mathbf{.}\\ \qquad\qquad (\ \overline{s} \lhd \mathsf{Accept}\langle\rangle \oplus \overline{s} \lhd \mathsf{Reject}\langle\rangle\ )\ ]\quad \mid\\ \mathsf{Seller}[\ \mathbf{!}\,\mathsf{QuoteCh}(s)\mathbf{.}\ \overline{s} \lhd \mathsf{Quote}\langle 300\rangle\mathbf{.}\\ \qquad\qquad s \rhd (\mathsf{Accept}()\ +\ \mathsf{Reject}())]\end{array}$$

Assuming we have a type for the specification, we can deduce, from the projection, α, the minimal type for QuoteCh, equal to

$$s \blacktriangleleft \mathsf{Quote}(\mathtt{int})\mathbf{.}\ \ s \blacktriangleright\ (\ \mathsf{Accept}(\mathtt{null})\ +\ \mathsf{Reject}(\mathtt{null})\ )$$

Notice that QuoteCh : $(s)\alpha$, even though it is not minimal, types the network System as well (its minimal type is instead obtained by adding an extra option to the branching corresponding to the operation Restart). This observation gives a hint that System is conform to the specification. In fact, this is true as all the options specified in the type are mimicked by the Spec (so the specification simulates the implementation).

In order to show that checking only the type is not enough, let us consider another system, say System2, where the buyer's behaviour is instead $P \mid P$. In this case, the network is still typed by QuoteCh : $(s)\alpha$ but, because of P occurring twice, System2 is not type-simulated by Spec and then not conform to the specification.

In summary, let I be a global description consisting of A and other participants. Suppose P is a program which implements A's behaviour. Then we can check the conformance of P against the specification I by projecting I to A, which we call S, and check P conforms to S; the relation "P conforms to S" can be taken as, for example, the converse of the weak similarity with respect to typed transitions under the minimal typing of S (for the formal definition, see [13]). We can use this notion via either hand-calculation (coinduction), model checking (e.g. mobility workbench), mechanical syntactic approximation, or as a basis of runtime monitoring.

7 Conclusions

This paper introduced a new formalism based on global description of communication behaviour, and the corresponding applied π-calculus. Both calculi are based on a new extension of session types, which can handle parallel interaction in one session. A theory of endpoint projection is developed, giving the three well-structuredness conditions on global descriptions. The sound and complete mapping from them to the corresponding endpoint processes is established.

Global descriptions have been practiced in various engineering contexts for a long time: the present work is a trial to realise its potential as a general programming method,

centring on type structures for communication and the end-point projection. The EPP theory needs be further explored for all basic concurrent programming primitives, including general sequencing, various mutual exclusion operations, exceptions, timeout and other useful primitives. While channel passing in our language can encode a synchronisation mechanism, a valuable future topic is its interaction with primitives for locking primitives and software transaction memory, since the notion of atomicity undergoes a fundamental change when we move to communication-centered programming.

References

1. Online appendix. http://www.dcs.qmul.ac.uk/~carbonem/cdlpaper.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, Jan. 1999.
3. R. Amadio, G. Boudol, and C. Lhoussaine. The receptive distributed pi-calculus. In *Proc. of the FST-TCS '99*, volume 1738 of *LNCS*. Springer-Verlag, 1999.
4. J. Baeten, H. van Beek, and S. Mauw. Specifying internet applications with DiCons. In *SAC '01*, pages 576–584, 2001.
5. N. Benton, L. Cardelli, and C. Fournet. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804, 2004.
6. M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, 2001.
7. M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. In *Proc. FOSSACS'03*, 2003.
8. K. Bhargavan, C. Fournet, and A. Gordon. Verified reference implementations of WS-Security protocols. *To appear in WS-FM '06*, 2006.
9. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW*, pages 82–96, 2001.
10. E. Bonelli, A. B. Compagnoni, and E. L. Gunter. Correspondence assertions for process synchronization in concurrent communications. *Journal of Functional Programming*, 15(2):219–247, 2005.
11. N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. Choreography and orchestration conformance for system design. In *COORDINATION*, volume 4038 of *LNCS*, pages 63–81, 2006.
12. M. Carbone, K. Honda, and N. Yoshida. Theoretical basis of communication-centred concurrent programming (part one). http://www.dcs.qmul.ac.uk/~carbonem/cdlpaper, November 2005.
13. 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.
14. M. Dezani-Ciancaglini, D. Mostrous, N. Yoshida, and S. Drossopoulou. Session Types for Object-Oriented Languages. In *Proceedings of ECOOP'06*, LNCS, 2006.
15. C. Fournet, C. A. R. Hoare, S. K. Rajamani, and J. Rehof. Stuck-free conformance. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2004.
16. S. Gay and M. Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, Nov. 2005.
17. K. Honda. Composing processes. In *Proceedings of POPL'96*, pages 344–357, 1996.
18. K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *ESOP '98*, pages 122–138. Springer, 1998.

19. K. Honda, N. Yoshida, and M. Berger. Control in the π-calculus. In *Proc. Fourth ACM-SIGPLAN Continuation Workshop (CW'04)*, 2004.
20. A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. In *POPL*, pages 128–141, 2001.
21. International Telecommunication Union. Recommendation Z.120: Message sequence chart, 1996.
22. N. Kobayashi, B. Pierce, and D. Turner. Linear types and π-calculus. In *Proceedings of POPL'96*, pages 358–371, 1996.
23. C. Laneve and L. Padovani. Smooth orchestrators. In *FoSSaCS '06*, LNCS, pages 32–46, 2006.
24. R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.
25. 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.
26. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
27. OMG. Unified modelling language, version 2.0, 2004.
28. PI4SOA. http://www.pi4soa.org.
29. B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.
30. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, Oct. 1996.
31. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.
32. J. Rehof. Lacking. In *POPL*, 2004.
33. J. Rehof. Lacking. In *POPL*, 2004.
34. S. Ross-Talbot and T. Fletcher. Ws-cdl primer. Unpublished draft, May 2006.
35. D. Sangiorgi. Uniform receptive. In *ICALP*, 2004.
36. D. Sangiorgi. Modal theory. In *ICALP*, 2005.
37. K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413, 1994.
38. W. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Information Technology and Management Journal*, 2(3):195–231, 2002.
39. V. T. Vasconcelos, A. Ravara, and S. J. Gay. Session types for functional multithreading. In *CONCUR '04*, LNCS, pages 497–511, 2004.
40. W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.
41. N. Yoshida, M. Berger, and K. Honda. Strong Normalisation in the π-Calculus. In *Proc. LICS'01*, pages 311–322. IEEE, 2001. The full version to appear in *Journal of Inf. & Comp.*.

A An Example of Endpoint Projection

In the following we illustrate the formal notion of endpoint projection we have developed in the paper using a fairly large toy example involving five participants. First, we explain the example in English; then we introduce the description in the global calculus; finally we project the description to endpoint processes.

A.1 Global Description in English

The example is an extension of the buyer-seller example introduced in section 2. The participants involved in this protocol are

1. Buyer (B)
2. Seller (S)
3. Vendor (V)
4. CreditChecker (CC)
5. RoyalMail (RM)

The protocol proceeds as follows:

1. Buyer requests a service ch_{CC} for company check to the credit checker CreditChecker by sending its name.
2. At this point CreditChecker can either give a positive or negative answer.
3. If the answer is positive:
   (a) Buyer asks Seller for a quote about product prod;
   (b) Seller then asks Vendor for service ch_V
   (c) Seller starts recursion and asks Vendor for a quote about product prod;
   (d) Vendor replies with a quote quote;
   (e) Seller forwards quote to Buyer increasing it by 10 units (quote+10);
   (f) if the quote is reasonable ($reasonable$(quote + 10)) then:
      i. Buyer sends Seller a confirmation (quoteOK) together with the credit (cred);
      ii. Seller then contacts CreditChecker for checking the credit;
      iii. If the credit is good then:
         A. Seller contacts Shipper (service ch_{Sh});
         B. Seller sends the delivery address;
         C. Shipper sends a confirmation;
         D. Seller forwards confirmation to Buyer;
      iv. If the credit is bad:
         A. CreditChecker tells Buyer;
         B. Buyer tells Seller terminating the protocol;
   (g) if the quote is not reasonable the protocol goes back to point 3c;
4. If the answer is negative then the protocol terminates.

A.2 Global Description in the Calculus

The global description consists of several components for readability. We directly give annotated interaction. The main description is:

1. $B^1 \to CC^2 : ch_{CC}(\nu\, s)$**.** $CC^2 \to B^1 : s\langle\mathsf{ack}\rangle$**.**
2. $B^1 \to CC^2 : s\langle\mathsf{companyCheck},\ \mathsf{sellerName},\ \mathsf{compName}\rangle$**.**
3. {
4. $\quad CC^2 \to B^1 : s\langle\mathsf{good}\rangle$**.** I_{good}
5. $\quad +$
6. $\quad CC^2 \to B^1 : s\langle\mathsf{bad}\rangle$**.** $\mathbf{0}$
7. }

where I_{good} in Line 4 is:

1. $B^1 \to S^3 : ch_S(\nu\, t)$**.** $S^3 \to B^1 : r\langle\mathsf{ack}\rangle$**.**
2. $B^1 \to S^3 : t\langle\mathsf{quoteReq},\ \mathsf{prod},\ \mathsf{prod}\rangle$**.**
3. $S^3 \to V^4 : ch_V(\nu\, r)$**.**
4. $V^4 \to S^3 : r\langle\mathsf{ack}\rangle$**.**
5. μX^3**.** {
6. $\quad S^3 \to V^4 : r\langle\mathsf{quoteReq},\ \mathsf{prod},\ \mathsf{prod}\rangle$**.**
7. $\quad V^4 \to S^3 : r\langle\mathsf{quoteRes},\ \mathsf{quote},\ \mathsf{quote}\rangle$**.**
8. $\quad S^3 \to B^1 : t\langle\mathsf{quoteRes},\ \mathsf{quote} + 10,\ \mathsf{quote}\rangle$**.**
9. $\quad$ if $reasonable(\mathsf{quote})@B^1$ then
10. $\quad\quad B^1 \to S^3 : t\langle\mathsf{quoteOK},\ \mathsf{cred},\ \mathsf{cred}\rangle$**.**
11. $\quad\quad S^3 \to CC^5 : ch_{CC}(\nu\, u)$**.**
12. $\quad\quad CC^5 \to S^3 : u\langle\mathsf{ack}\rangle$**.**
13. $\quad\quad S^3 \to CC^5 : u\langle\mathsf{personalCreditCheck},\ \mathsf{cred{:}adr},\ \mathsf{cred{:}adr}\rangle$**.**
14. $\quad\quad$ {
15. $\quad\quad\quad CC^5 \to S^3 : u\langle\mathsf{good}\rangle$**.** I'_{good}
16. $\quad\quad\quad +$
17. $\quad\quad\quad CC^5 \to S^3 : u\langle\mathsf{bad}\rangle$**.**
18. $\quad\quad\quad S^3 \to B^1 : t\langle\mathsf{yourCreditIsBad}\rangle$**.** $\mathbf{0}$
19. $\quad\quad$ }
20. $\quad$ else $B^1 \to S^3 : t\langle\mathsf{quoteNotOK}\rangle$**.** X^3
21. }

where I'_{good} in Line 15 is:

1. $S^3 \to R^6 : ch_R(\nu\, p)$**.**
2. $R^6 \to S^3 : p\langle\mathsf{ack}\rangle$**.**
3. $S^3 \to R^6 : p\langle\mathsf{deliv},\ \mathsf{adr},\ \mathsf{adr}\rangle$**.**
4. $R^6 \to S^3 : p\langle\mathsf{conf}\rangle$**.**
5. $S^3 \to B^1 : t\langle\mathsf{conf}\rangle$**.** $\mathbf{0}$

We can check these descriptions are typable, strongly connected, well-threaded and coherent. For connectedness, the descritption given above uses a lot of acks. As we discussed in the long version, many of these acks are in fact unnecessary by using a relaxed notion of connectedness.

A.3 End-Point Projection of the Global Interaction

Following the definition of EPP in the paper, we first project the global descrtiption onto each thread. The first one is Buyer's only thread.

$$
\begin{aligned}
\mathsf{TP}(I,1) = \overline{ch_{CC}}(\nu s).\ & s \triangleright \mathsf{ack}().\ \overline{s} \triangleleft \mathsf{companyCheck}\langle \mathsf{sellerName}\rangle. \\
& \{\ s \triangleright \mathsf{good}().\ \overline{ch_S}(\nu t).\ t \triangleright \mathsf{ack}().\ \overline{t} \triangleleft \mathsf{quoteReq}\langle \mathsf{prod}\rangle. \\
& \ \mu X.\ \ t \triangleright \mathsf{quoteRes}(\mathsf{quote}). \\
& \quad \mathsf{if}\ \mathit{reasonable}\{\mathsf{quote}\}\ \mathsf{then}\ \overline{t} \triangleleft \mathsf{quoteOK}\langle \mathsf{cred}\rangle. \\
& \qquad\qquad\qquad\qquad \{t \triangleright \mathsf{yourCreditIsBad}() \\
& \qquad\qquad\qquad\qquad + \\
& \qquad\qquad\qquad\qquad t \triangleright \mathsf{conf}()\} \\
& \quad \mathsf{else}\ \overline{t} \triangleleft \mathsf{quoteNoteOK}\langle\rangle.\ X \\
& + \\
& \ s \triangleright \mathsf{bad}().\ \mathbf{0}\ \ \}
\end{aligned}
$$

Note this thread starts before the recursion and go through inside the (global) recursion. Thus the projected endpoint behaviour also contains recursion.

The next projection is onto the first thread of CreditChecker (note this participant has two threads, 2 and 5).

$$
\begin{aligned}
\mathsf{TP}(I,2) = \mathbf{!}\, ch_{CC}(s).\ & \overline{s} \triangleleft \mathsf{ack}\langle\rangle.\ s \triangleright \mathsf{companyCheck}(\mathsf{compName}). \\
& \{\ \overline{s} \triangleleft \mathsf{good}\langle\rangle. \\
& \ \oplus \\
& \ \overline{s} \triangleleft \mathsf{bad}\langle\rangle.\ \mathbf{0}\ \ \}
\end{aligned}
$$

Note no recursion is involved in this thread projection, simply because the thread 2 does not occur inside the recursion.

Next we jump to Thread 5, which is another component of CreditChecker.

$$
\begin{aligned}
\mathsf{TP}(I,5) = \mathbf{!}\, ch_{CC}(u).\ & \overline{u} \triangleleft \mathsf{ack}\langle\rangle.\ u \triangleright \mathsf{personalCreditCheck}(\mathsf{cred{:}adr}) \\
& (\overline{u} \triangleleft \mathsf{good}\langle\rangle \\
& \ + \\
& \ \overline{u} \triangleleft \mathsf{bad}\langle\rangle
\end{aligned}
$$

Note the process does not include the recursion either. This is because it is inside a recursion and it initiates a new thread there. As a result the code is identical with the projection onto Thread 2.

We now move to the projection onto the unique thread of Seller, which is Thread 3.

$$\begin{aligned}
\mathsf{TP}(I,3) = {} & !\, ch_S(t).\ \overline{t} \lhd \mathsf{ack}\langle\rangle.\ t \rhd \mathsf{quoteReq(prod)}.\ \overline{ch_V}(\nu r).\ t \rhd \mathsf{ack}().\\
& \mu X.\ \ \overline{r} \lhd \mathsf{quoteReq}\langle\mathsf{prod}\rangle.\ r \rhd \mathsf{quoteRes(quote)}.\\
& \quad \overline{t} \lhd \mathsf{quoteRes}\langle\mathsf{quote} + 10\rangle.\\
& \quad \{\ t \rhd \mathsf{quoteOK(cred)}.\ \overline{ch_C C}(\nu u).\ u \rhd \mathsf{ack}().\\
& \quad\ \overline{u} \lhd \mathsf{personalCreditCheck}\langle\mathsf{cred{:}adr}\rangle.\\
& \quad\quad \{u \rhd \mathsf{good}().\ \overline{ch_R}(\nu p).\ p \rhd \mathsf{ack}()\\
& \quad\quad\ \overline{p} \lhd \mathsf{deliv}\langle\mathsf{adr}\rangle.\ p \rhd \mathsf{conf}()\overline{t} \lhd \mathsf{conf}\langle\rangle\\
& \quad\quad\ +\\
& \quad\quad\ u \rhd \mathsf{bad}().\ \overline{t} \lhd \mathsf{CreditIsBad}\langle\rangle\ \}\\
& \quad\ +\\
& \quad\ t \rhd \mathsf{quoteNoteOK}().\ X
\end{aligned}$$

As before, this thread starts outside of the recursion in the global description and is also used inside, so that both the recursion and the recursion variable are used as they are, leading to the recursive behaviour of the process. Note how the use of session functions as a way to handle recursion appropriately in EPP.

The projection onto the unique thread of Vendor follows.

$$\begin{aligned}
\mathsf{TP}(I,4) = {} & !\, ch_V(r).\ \overline{t} \lhd \mathsf{ack}\langle\rangle.\\
& \mu X.\ r \rhd \mathsf{quoteReq(prod)}.\ \overline{r} \lhd \mathsf{QuoteRes}\langle\mathsf{quote}\rangle.\ X
\end{aligned}$$

Finally we end with the projection onto Thread 6, giving the simple behaviour of RoyalMail.

$$\mathsf{TP}(I,6) = !\, ch_R(p).\ \overline{p} \lhd \mathsf{ack}\langle\rangle.\ p \rhd \mathsf{deliv(adr)}.\ \overline{p} \lhd \mathsf{conf}\langle\rangle$$

As before, Thread 6 does noont contain recursion since it is fully inside the (global) recursion, initiating a thread there.

As noted, there are two threads (2 and 5) that belong to the same class of equivalence i.e. they are part of the same service channel ch_{CC}. This means that we must merge the two threads in the final EPP. By applying the merge operator, and noting they are evidently mergeable, we get the following process:

$$\begin{aligned}
& !\, ch_{CC}(u).\ \overline{u} \lhd \mathsf{ack}\langle\rangle.\\
& \quad u \rhd \left(\begin{array}{l}
\mathsf{personalCreditCheck(cred{:}adr)}.\ (\overline{u} \lhd \mathsf{good}\langle\rangle \oplus \mathsf{bad}\langle\rangle)\\
\qquad\qquad +\\
\mathsf{companyCheck(compName)}.\ (\overline{u} \lhd \mathsf{good}\langle\rangle \oplus \mathsf{bad}\langle\rangle)
\end{array}\right)
\end{aligned}$$

By which we have arrived at the endpoint behaviours of all participants realising the original global description.

The projection works because of the linear usage of channels inside each session and service channel principle, as well as the three well-structuredness conditions. We

believe many business protocols conform to these conditions (modulo relaxation of connectedness we discussed in the long version). How these conditions can be extended in disciplined ways to allow more "untamed" protocols (such as those involving exceptions) to be treated in the theory, is an interesting subject of further studies.