Structured Communication-Centred Programming for Web Services

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

[1] Department of Computing, Imperial College London
[2] Department of Computer Science, Queen Mary University of London

Abstract. This paper relates two different paradigms of descriptions of communication behaviour, one focussing on global message flows and another on end-point behaviours, using formal calculi based on session types. The global calculus, which originates from a web service description language (W3C WS-CDL), describes an interaction scenario from a vantage viewpoint; the end-point calculus, an applied typed π-calculus, precisely identifies a local behaviour of each participant. We explore a theory of end-point projection, by which we can map a global description to its end-point counterpart preserving types and dynamics. Three principles of well-structured description and the type structures play a fundamental role in the theory.

1 Introduction

Communication-Centred Programming. The explosive growth of Internet in the last decades has led to the de facto, global standards for naming scheme (URI, Domain Names), communication protocols (SOAP, HTTP, TCP/IP) and message format (XML). These elements offer a useful basis for building applications centring on communication among distributed agents through these standards. Such communication-centred applications are sometimes called *web services*. Web services are an active area of infrastructural development, involving the major standardisation bodies such as W3C and OASIS.

A concrete application area of communication-centred applications is *business protocol*. A business protocol is a series of structured and automated interactions among business entities. It is predominantly inter-domain, is often regulation-bound, and demands clear shared understanding about its meaning. Some protocols such as industry standards will remain unchanged for a long time once specified; others may undergo frequent updates. Because of its inherent inter-organisational nature, there is a strong demand for a common standard for specifying business protocols on a sound technical basis.

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) [26], developed by W3C WS-CDL Working Group since 2004 in collaboration with π-calculus experts including the present authors. WS-CDL offers a fully expressive global description language for channel-based

R. De Nicola (Ed.): ESOP 2007, LNCS 4421, pp. 2–17, 2007.

communication equipped with general control constructs (e.g. sequencing, conditionals and recursion), and is conceived with potential usage of type-based formal validation from the outset. The intuition behind the term *choreography* may be summarised thus:

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

WS-CDL is conceived as a language for describing such a "global scenario": once specified, this scenario is to be executed by individual distributed processes without a single point of control.[1] Another significant feature is WS-CDL's informal use of *sessions* for communication: at the outset of each run of a protocol, a session is established between communication parties so that involved communications can be distinguished from different runs of the same or other protocols.

End-Point Projection. The global description of a communication behaviour is useful since, among others, it offers a clear view of dynamics of the whole interactions. Real execution of the description, however, is always through communication among distributed end-points 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 end-point processes so that their interactions precisely realise the original global description?*

Such a projection may be called *end-point projection (EPP)*, a terminology from WS-CDL Working Group. Having a universally agreed and well-founded EPP is fundamental for the engineering use of global descriptions, from design to implementations to validations/verifications to run-time monitoring (see § 5).

This paper establishes a formal theory of EPP by introducing the two typed calculi for interaction, a distilled version of WS-CDL (a global calculus) and an applied π-calculus (an end-point calculus), and defining a mapping from the former to the latter. This mapping is highly non-trivial due to the different nature of descriptions: a global calculus directly describes interactions among multiple participants involving sequencing, branching and recursion, which differs from the end-point-based description given in the π-calculus. A central contribution of this work is the identification of three basic principles for global descriptions under which we can define a *sound* and *complete* EPP, in the sense that, through a given EPP, all and only globally described behaviour is realised as communication among end-points. The three principles are: *connectedness* (a basic local causality principle), *well-threadedness* (a stronger locality principle based on session types [23,16,12,25,13,6]) and *coherence* (a consistency principle for description of each participant in a global description). Schematically, the EPP mapping has the following shape:

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

[1] A related idea is *orchestration* where one master component, "conductor", directly controls activity of one or more slave components, which is useful when communicating parties can be placed under a common administrative domain, see [1].

where I is a global description, A, B and C are *participants* of the protocol and P, Q and R are projections of I onto A, B and C respectively. We shall show that, when applied to well-typed interactions following the three principles, the EPP mapping thus defined satisfies *type preservation*, *soundness* and *completeness*.

The EPP theory opens a conduit between global descriptions and accumulated studies on process calculi, allowing the use of the latter's rich theories for engineering aims. The EPP theory will be published as an associated document of WS-CDL 1.0 [11] (which contains many examples and full technical details), and will form part of its open-source implementation [19].

Related Work. Global methods for describing communication have been practiced in many different forms, including MSCs, UML diagrams and Petri-Nets [24]. In the context of security study, Strand Space [15] is a model for analysing security protocols based on their global representation; while Briais and Nestmann [7] present a notation for representing protocol narrations and relate it to the π-calculus (which is a form of end-point projection in our sense). These notations and models offer a useful basis for design/specification/analysis, but are not intended as full-fledged programming languages, so that they lack in e.g. general control structures and constructs for value passing and state change.

DiCons is a global notation for programming Internet applications [2] whose primitives include web server invocation, email, and web form filing. A formal notion of end-point projection has not been studied in [2].

The present work shares with many recent works its use of types of the mobile processes, including, but not limited to, Pict [20], Polyphonic C♯ [3] and the preceding studies on session type disciplines [6,12,13,16,23,25]. In the context of session types, our work extends their usage to global descriptions and intra-session parallel communications. These preceding works are based on end-point languages and calculi. The EPP theory offers a passage through which these and other related studies can be reflected onto global descriptions.

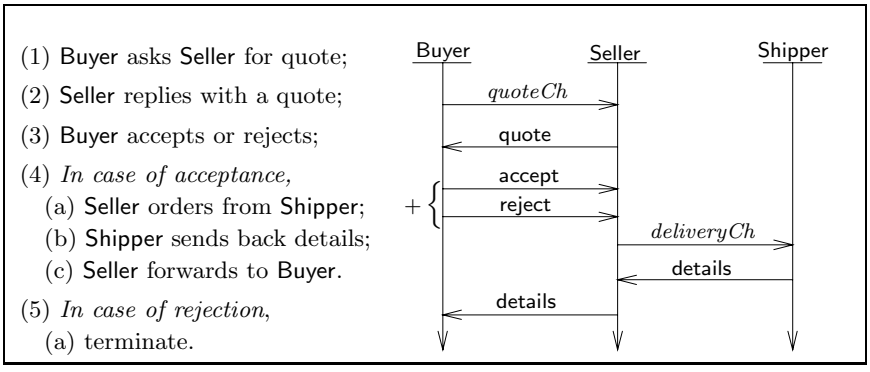
Fournet, Gordon, Bhargavan and Corin studied security-related aspects of web services. In [5], they have implemented part of WS-Security libraries, and analyse them through a translation into the π-calculus. The benefits of such a tool may be reflected onto global descriptions through the theory of EPP.

Laneve and Padovani [17] give a model of orchestrations using an extensions of π-calculus to join patterns. Busi et al. [8] study a bisimulation-based correspondence between choreography and orchestration. In [14], they further studied a calculus for web services of end-point descriptions based on predicate-driven communication. A formal theory of end-point projection is the main difference of our work from these preceding works.

2 The Global Calculus

2.1 Buyer-Seller Protocol

We outline the key technical ideas using an example from [21], the "Buyer-Seller Protocol". The participants involved are a Buyer, a Seller and a Shipper. We describe the protocol with both text and a sequence diagram.



The diagram is ambiguous at the branching (+) actions in (4) and (5): the purpose of such diagrams is to offer an informal overview: they naturally omit detailed control structures (choices, loops, etc.) and manipulation of values/states. The reason why such global descriptions are practised in engineering is because they enable a clear grasp of the whole interaction structure, lessening synchronisation and other errors at the design stage.

WS-CDL is intended to extend these virtues of global notations to a full fledged description language. We find, through our involvement in its design process, that it is based on two engineering principles: the **Service Channel Principle (SCP)** where invocation channels (e.g. a channel at which Buyer first communicates to Seller, or Seller to Shipper) can be shared and invoked repeatedly; and the **Session Principle (SP)** where a sequence of conversations belonging to a protocol should not be confused with other concurrent runs of this or other protocols by the participants i.e. each such sequence should form a logical unit of a conversation, or a *session*.

(**SCP**) corresponds to the repeated availability of replicated input channels in the π-calculus (called *uniformly receptive* [22] and *server channels* in [4]), or, in practice, of public URLs. (**SP**) is a basic principle in many communication-centred programs, and can be given simple type abstraction with decidable type checking [12,16,25].[2] The global calculus is built from formalisation of these two principles, as well as combinators for composing descriptions. Before introducing the syntax formally, we first outline its basic ideas using an example.

Figure 1 (a) gives a description of the Buyer-Seller Protocol in the global calculus. In (a), Line 1 describes Action (1) in the protocol. The *quoteCh* is a *service channel*, which may be considered as a public 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 channel*, a fresh name to 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 between two parties is directly described as one interaction.*

[2] In implementations of web services, sessions are implemented using so-called *correlation identities* (which may be considered as nonces in cryptographic protocols).

1. Buyer → Seller : $quoteCh(\boldsymbol{\nu}\, s)$.
2. Seller → Buyer : $s\langle$quote, 300, $x\rangle$. {
3. {Buyer → Seller : $s\langle$accept$\rangle$.
4. Seller → Shipper : $delivCh(\boldsymbol{\nu}\, t)$.
5. Shipper → Seller : $t\langle$details, v, $x\rangle$.
6. Seller → Buyer : $s\langle$details, x, $y\rangle$. $\mathbf{0}$ }
7. +
8. {Buyer → Seller : $s\langle$reject$\rangle$. $\mathbf{0}$} }

(a) Protocol for Buyer-Seller Example

1. Buyer → Seller : $quoteCh(\boldsymbol{\nu}\, s)$.
2. rec X. {
3. Seller → Buyer : $s\langle$quote, q, $x\rangle$.
4. if `reasonable`(x)@Buyer then
5. {Buyer → Seller : $s\langle$accept$\rangle$.
6. Seller → Shipper : $delivCh(\boldsymbol{\nu}\, t)$.
7. Shipper → Seller : $t\langle$details, v, $x\rangle$.
8. Seller → Buyer : $s\langle$details, x, $y\rangle$. $\mathbf{0}$ }
9. else
10. {Buyer → Seller : $s\langle$reject$\rangle$.
q@Seller $:= q$@Seller $- 1$. X } }

(b) Protocol with Recursion

Fig. 1. Business Protocols in the Global Calculus

Line 2 describes Action (2), Seller's reply to Buyer. The session has already been started and now the two participants communicate using the session channel s. In addition, three factors involved: quote identifies the particular operation used in this communication (i.e. request for 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 channel t (Line 4). Then in Line 5, Shipper sends back the shipping details through t. Finally in Line 6, Seller forwards the details to Buyer by sending the value stored in variable x: here the protocol terminates. In Line 8, Buyer communicates reject, in which case the protocol immediately terminates.

In (a), we can observe the distinction between service channels and session channels implements (**SCP**) and (**SP**); sessions offer logical grouping of threads of interactions, where each thread starts with a procedure-call-like service invocation at a service channel and carry out in-session communications at associated session channels. This point can be seen more clearly in Fig. 1 (b), a refinement of (a). In (b), if Buyer chooses reject, the protocol recurs to Line 3, after decrementing the quote. In Line 4, a unary predicate `reasonable`(x) is evaluated at Seller's site ("@" indicates a location, similarly in Line 10). The session notation makes it clear that all quote-messages from Seller to Buyer in the recursion are done within a single session. §4 shall show that such session information plays a crucial role in tractable end-point projection.

2.2 Syntax and Dynamics

The syntax of the global calculus [9] is given by BNF. $I, I', \ldots$ denote *terms* of the calculus, also called *interactions*. $ch, ch' \ldots$ range over *service channels*; $s, t, \ldots$ range over *session channels*; $\tilde{s}$ indicates a vector of session channels; $A, B, C, \ldots$ range over *participants*; $x, y, z, \ldots$ over variables local to each participant; $X, X', \ldots$ over *term variables*; and $e, e', \ldots$ over arithmetic and other first-order expressions.

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

(init) denotes a session initiation by A via B's service channel ch, with fresh session channels $\tilde{s}$ and continuation I. (comm) denotes an in-session communication over a session channel s, where op is an operator name. Note that y does not bind in I. "|" and "+" denote respectively parallel and choice. $(\boldsymbol{\nu} s)\ I$ is the π-calculus-like name restriction, binding s in I. Since such a hiding is only generated by session initiation, we stipulate that a hiding never occurs inside a prefix, sum or conditional. (cond) and (assign) are 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. We often omit $\mathbf{0}$ and empty vectors.

The reduction of the global calculus is 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 σ' which is identical to σ except that $\sigma'@A(y)$ is equal to v. A reduction "$(\sigma,\ I) \to (\sigma',\ I')$" says that I in the state σ performs one-step computation and becomes I' with the new state σ'. Below we list some of the rules generating the reduction (a complete set of rules can be found in [11]).

$$
\text{(G-Init)}\ (\sigma,\ A \to B : ch(\boldsymbol{\nu}\, \tilde{s}).\ I) \to (\sigma, (\boldsymbol{\nu} \tilde{s})\ I)
$$

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

(G-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 $(\boldsymbol{\nu} \tilde{s})$), and the next I is unfolded. The initiation channel ch will play an important role for typing and the end-point projection later. (G-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 the new state $\sigma[x@B \mapsto v]$. The same variable (say x) located at different participants are distinct (hence $\sigma@A(x)$ and $\sigma@B(x)$ may differ). Other rules for parallel, summation, recursion and restriction are omitted.

As an example of reduction, consider, for instance:

$$
\mathsf{Buyer} \to \mathsf{Seller} : \mathit{quoteCh}(\boldsymbol{\nu}\, s).\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{quote},\ 300,\ x\rangle.\ I'
$$

with state σ. By (G-Init), we get $(\sigma, (\boldsymbol{\nu} s)\ \mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{quote},\ 300,\ x\rangle.\ I')$. Now, by rule (G-Com), this evolves into $(\sigma[x@\mathsf{Buyer} \mapsto 300], (\boldsymbol{\nu} s)\ I')$.

2.3 Session Types for Global Descriptions

We use a generalisation of session types [16]. The grammar of types follows.

$$\alpha \ ::= \ \ s \blacktriangleright \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i \mid \ \ s \blacktriangleleft \Sigma_i \mathsf{op_i}(\theta_i).\ \alpha_i \mid \ \ \alpha_1 \mid \alpha_2 \mid \ \ \mathtt{end} \mid \ \ \mu \mathbf{t}.\ \alpha \mid \ \ \mathbf{t}$$

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 a process is ready to receive any of the (pairwise distinct) operators $\{\mathsf{op_i}\}$, each 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 its exact dual. 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 $\mathtt{end}$, the *inaction type* indicating session termination, being the identity. We demand session channels in α_1 and α_2 to be disjoint: this guarantees 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 [13].

Note that session channels occur free in session types: this is necessary to allow multiple session channels to be used in parallel in a single session; with this, we can faithfully capture use cases of web services which exchange different data simultaneously, leading to a generalisation of session types in the literature. Let us show a simple example:

$$s \blacktriangleleft \mathsf{quote}(\mathtt{int}).\ \mathtt{end} \mid s' \blacktriangleleft \mathsf{extra}(\mathtt{string}).\ \mathtt{end}$$

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 and result in a type error.

A *typing judgment* has the form $\Gamma \vdash I : \Delta$ where Γ is *service typing* and Δ *session typing*. The grammar of typings follows where $A \neq B$ in $\tilde{s}[A, B]$:

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

Each time a session is initiated, session channels need be freshly generated. Thus, the type of a service channel indicates a vector of session channels to be initially exchanged, in addition to how they are used. This is formulated by *service type* $(\tilde{s})\alpha$ 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. In a service typing, $ch@A : (\tilde{s})\alpha$ says that ch is located at A and offers a service interface $(\tilde{s})\alpha$; $x@A : \theta$ says that a variable x located at A may store values of type θ; finally, $X : \Delta$ says that when the interaction recurs to X, it should have the typing Δ.

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

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

$$\text{(G-TCom}_2\text{)}\ \frac{\Gamma \vdash I \triangleright \Delta, \tilde{s}\,[B,A]:\alpha_j \quad \Gamma \vdash e@A:\theta_j \quad \Gamma \vdash x@B:\theta_j \quad s \in \{\tilde{s}\} \quad j \in J}{\Gamma \ \vdash\ A \to B : s\langle \mathsf{op_j},\ e,\ x\rangle.\ I \ \triangleright\ \Delta, \tilde{s}\,[B,A]:s \blacktriangleright \Sigma_{i\in J}\mathsf{op_i}(\theta_i).\ \alpha_i}$$

$$\text{(G-TPar)}\ \frac{\Gamma \vdash I_1 \triangleright \Delta_1 \quad \Gamma \vdash I_2 \triangleright \Delta_2}{\Gamma \vdash I_1 \mid I_2 \triangleright \Delta_1 \bullet \Delta_2} \qquad \text{(G-TInit)}\ \frac{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash I \triangleright \Delta, \tilde{s}\,[B,A]:\alpha}{\Gamma,\ ch@B:(\tilde{s})\alpha \vdash A \to B : ch(\boldsymbol{\nu}\,\tilde{s}).\ I \triangleright \Delta}$$

Rule (G-TCom) states that, for typing an in-session communication of e from A to B at s with the choice op_j, (1) the body I should assign α_j to $\tilde{s}$ containing s; (2) the value e should be typed in the source (A) with θ_j; and (3) the variable (parameter) x should be typed in the target (B) with the same type. Then, in the conclusion, a branching type is formed whose j-th branch consists of op_j, θ_j and α_i. In (G-TCom), the session type in focus is considered direction from the viewpoint of A. We may also regard it from the receiver's viewpoint (B), which is its symmetric variant (G-TCom$_2$). Rule (G-TPar) uses the linearity condition found in [16]. The the operator $\bullet$ is well-defined whenever the linearity condition is satisfied and is such that $\tilde{s}[A,B] : \alpha \in \Delta_1 \bullet \Delta_2$ iff either $\tilde{s}[A,B] : \alpha_1 \in \Delta_1$, $\tilde{s}[A,B] : \alpha_2 \in \Delta_2$ and $\alpha = \alpha_1 \mid \alpha_2$; or $\tilde{s}[A,B] : \alpha \in \Delta_1$ and $\{\tilde{s}\} \cap \mathsf{fsc}(\Delta_2) = \emptyset$; or its symmetric case. The other rules are standard [11].

As a simple example, we type the Buyer-Seller interaction I in Fig. 1 (a). Service channel *quoteCh* is assigned with the following service type:

$$(s)\ s \blacktriangleleft \mathsf{quote}(\mathtt{integer}).\ s \blacktriangleright (\ \mathsf{accept}(\mathtt{null}).\ s \blacktriangleleft \mathsf{details}(\mathtt{string}).\ \mathtt{end} + \mathsf{reject}(\mathtt{null}).\ \mathtt{end}\ )$$

Service channel *deliveryCh* has type $(t)\ t \blacktriangleleft \mathsf{details}(\mathtt{string}).\ \mathtt{end}$. Denoting two types by $(s)\alpha_1$ and $(t)\alpha_2$, we have: $\mathit{quoteCh}:(s)\,\alpha_1,\ \mathit{deliveryCh}:(t)\,\alpha_2 \ \vdash\ I \triangleright \emptyset$.

Similarly, we can type the interaction in Figure 1 (b) where we have recursion. The typing of the service channel *quoteCh* will differ in the "rejection" branch, given as: $(s)\ \mu\mathbf{t}.\ s \blacktriangleleft \mathsf{quote}(\mathtt{integer}).\ s \blacktriangleright (\ldots + \mathsf{reject}(\mathtt{null}).\ \mathbf{t})$.

The typing system also incorporates subtyping based on an inclusion ordering on each type (formalised using simulation like in [13]).

Theorem 1 (Subject Reduction). *Assume $\Gamma \vdash \sigma$. Then $\Gamma \ \vdash\ I \triangleright \Delta$ and $(\sigma, I) \to (\sigma', I')$ imply $\Gamma \vdash \sigma'$ and $\Gamma \ \vdash\ I' \triangleright \Delta'$ for some Δ' s.t. $\mathsf{fsc}(\Delta') \subset \mathsf{fsc}(\Delta)$.*

3 The End-Point Calculus

3.1 Syntax and Dynamics

The end-point calculus is the π-calculus [18] extended with sessions [16] as well as locations and store [10]. $P, Q, \ldots$ denote *processes*, $M, N, \ldots$ *networks*.

$$\begin{array}{rcl} P & ::= & \mathbf{!}\,ch(\tilde{s}).\ P \mid \overline{ch}(\boldsymbol{\nu}\tilde{s}).\ P \mid s \triangleright \Sigma_i \mathsf{op}_i(y_i).\ P_i \mid \overline{s} \triangleleft \mathsf{op}\langle e\rangle.\ P \mid x := e.\ P \\ & & \mid \mathsf{if}\ e\ \mathsf{then}\ P_1\ \mathsf{else}\ P_2 \mid P_1 \oplus P_2 \mid P_1 \mid P_2 \mid (\boldsymbol{\nu}s)\ P \mid X \mid \mu X.\ P \mid \mathbf{0} \\ N & ::= & A[\,P\,]_\sigma \quad \mid \quad N_1 \mid N_2 \quad \mid \quad (\boldsymbol{\nu}s)\ N \quad \mid \quad \boldsymbol{\epsilon} \end{array}$$

The first two processes describe session initiations; the next two, in-session communications (where y_i in the first construct, branching input, is *not* bound in P_i, and $\{\mathsf{op}_i\}$ should be pairwise distinct). Next, $x := e.\ P$ assigns a value v to x in its store then continues as P. The rest is standard. Networks are parallel composition of participants, where a *participant* is of the shape $A[\ P\ ]_\sigma$, with A being the name of the participant, P its behaviour, and σ its local state. We often omit σ when irrelevant.

The reduction semantics for the end-point calculus follows the π-calculus. Below we list the three key rules (other rules are found in [11]).

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

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

$$\text{(E-Asgn)}\ A[\ x := e.\ P \mid P'\ ]_\sigma \to A[\ P \mid P'\ ]_{\sigma[x \mapsto v]} \quad (\sigma \vdash e \Downarrow v)$$

(E-Init) defines the session initiation: two participants A and B will synchronise to start a session, $!\,ch(\tilde{s}).\ P$ denoting a service and $\overline{ch}(\boldsymbol{\nu}\tilde{s}).\ Q$ a request. It will result in sharing fresh session names $\tilde{s}$ local to A and B. These session names are then used in (E-Com) for communication. In (E-Com), communicated values are assigned to local variables, rather than substituted, for having the correspondence with the global calculus. (E-Asgn) updates a local store.

3.2 Session Typing of End-Point Calculus

In the end-point calculus, we use two typing judgements, $\Gamma \vdash_A P \triangleright \Delta$ (where P is typed as a behaviour for A) and $\Gamma \vdash M \triangleright \Delta$. Γ (service typing) and Δ (session typing) are given as before except (1) Γ adds $\overline{ch}@A : (\tilde{s})\alpha$; and (2) we replace $\tilde{s}[A, B] : \alpha$ by $\tilde{s}@A : \alpha$. The selected typing rules are given below.

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

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

$$\text{(E-TServ)}\ \frac{\Gamma \vdash_A P \triangleright \tilde{s}@A : \alpha}{\Gamma,\ ch@A : (\tilde{s})\alpha \vdash_A !\,ch(\tilde{s}).\ P \triangleright \emptyset} \qquad \text{(E-TReq)}\ \frac{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash_A P \triangleright \Delta \cdot \tilde{s}@A : \alpha}{\Gamma,\ \overline{ch}@B : (\tilde{s})\alpha \vdash_A \overline{ch}(\boldsymbol{\nu}\tilde{s}).P \triangleright \Delta}$$

(E-TB) is for branching input. The resulting typing can have less branches than the real process, so that the process is prepared to receive any operator specified in the type. (E-TS) is its dual: the typing can have more branches than the real process, so that the process invokes at most those operators specified in the typing. Combining (E-TB) and (E-TS), an output never invokes a non-existent option in the input. (E-TServ) is for the server side of initialisation. In the premise, the session typing should not have session channels other than the target of initialisation: this prevents *free* session channels from occurring under the replicated input, thus guaranteeing their linear usage. By our

convention, neither ch nor $\overline{ch}$ occurs in Γ in the conclusion. The output side of initialisation (E-TREQ) is analogous, except it does not need the linearity constraint. The remaining rules are standard [16]: for example, with parallel composition, we ensure that an input of type α is composed with an output of its dual.

We recall our running example, Figure 1 (a) in § 2.1. An end-point representation of this example for Buyer may be written:

$$\mathsf{Buyer}[\ \overline{\mathit{quoteCh}}(\boldsymbol{\nu} s).\ s \triangleright \mathsf{quote}(x).\ (\overline{s} \triangleleft \mathsf{accept}.\ s \triangleright \mathsf{details}(y).\ \mathbf{0} \oplus \overline{s} \triangleleft \mathsf{reject}.\ \mathbf{0})\ ]$$

Above $\mathsf{Buyer}[P]$ indicates a participant (a named agent) whose behaviour is given by the process P. The Seller's code is given as:

$$\begin{array}{l}\mathsf{Seller}[\mathbf{!}\ \mathit{quoteCh}(s).\ \overline{s} \triangleleft \mathsf{quote}\langle 300\rangle.\ \ s \triangleright \\ \qquad (\mathsf{accept}.\ \overline{\mathit{deliveryCh}}(\boldsymbol{\nu} t).\ t \triangleright \mathsf{delivery}(x).\ \overline{s} \triangleleft \mathsf{delivery}\langle x\rangle.\ \mathbf{0}\ +\ \mathsf{reject}.\ \mathbf{0})]\end{array}$$

The end-point representation for Shipper is given similarly. These end-point descriptions do not directly and explicitly describe how interaction proceeds globally, which may often be the central concern of communication-centred applications designers/users. However, they precisely represent local communication behaviours which give rise to global interactions. The two service channels *quoteCh* and *deliveryCh* are replicated and ready to be invoked, following (**SCP**).

We can type these processes using the service types $(s)\alpha_1$ and $(t)\alpha_2$ from § 3.3. The type of the seller becomes (writing P for its process):

$$\mathit{quoteCh}\colon (s)\,\alpha_1,\ \overline{\mathit{deliveryCh}}\colon (t)\,\overline{\alpha}_2\ \vdash\ \mathsf{Seller}[\ P\ ]_\sigma\ \triangleright\ \emptyset.$$

Note that the service channel *deliveryCh* is overlined, indicating the direction: this is because the input channel is located at the shipper's. In the global calculus, a channel is always used for both input and output, so there is no such need. Similarly we may type the end-point processes for Buyer and Seller with recursion as in Figure 1 (b), as:

$$\begin{array}{l}\mathsf{Buyer}[\ \mu X.\ \overline{\mathit{quoteCh}}(\boldsymbol{\nu} s).\ s \triangleright \mathsf{quote}(x). \\ \qquad \mathsf{if}\ \mathsf{reasonable}(x)\ \mathsf{then}\ \overline{s} \triangleleft \mathsf{accept}.\ s \triangleright \mathsf{details}(y).\ \mathbf{0}\ \mathsf{else}\ \overline{s} \triangleleft \mathsf{reject}.\ X\ ] \quad | \\ \mathsf{Seller}[\ \mathbf{!}\ \mathit{quoteCh}(s).\ \mu X.\ \overline{s} \triangleleft \mathsf{quote}\langle 300\rangle.\ \ s \triangleright \\ \qquad (\mathsf{accept}.\ \overline{\mathit{deliveryCh}}(\boldsymbol{\nu} t).\ t \triangleright \mathsf{delivery}(x).\ \overline{s} \triangleleft \mathsf{delivery}\langle x\rangle.\ \mathbf{0} + \mathsf{reject}.\ X)]\end{array}$$

We may also note, both in its term and in its typing, the end-point process for Shipper in Figure 1 (b) does not involve recursion, since its session is self-contained inside a recursion.

Theorem 2 (subject reduction). *If* $\Gamma \vdash N \triangleright \Delta$ *and* $N \to N'$ *then* $\Gamma \vdash N' \triangleright \Delta$.

A significant corollary of this result is the lack of communication error in the sense that typed processes never invoke missing operations and never communicate ill-typed values. This is fundamental for end-point processes since they describe inputs and outputs separately, unlike global descriptions.

4 The End-Point Projection

4.1 Three Principles for End-Point Projections

A theory of EPP assigns to global descriptions the precise and transparent operational content as communicating processes. This task becomes subtle because a *global calculus allows descriptions that do not make sense at end-points, i.e. as distributed communicating processes.* Below we discuss three issues in this regard one by one, together with the corresponding disciplines which disallow them.

Connectedness. Consider the following code snippet for global description.

$$\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s).\ \ \mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t)$$

Remembering "." indicates sequencing, Shipper is described as contacting Depot only after Buyer has performed a request to Seller in the description above. Implementing this behaviour as distributed processes demands that Shipper be notified once the first communication is performed by message passing, for instance in:

$$\mathsf{Buyer} \to \mathsf{Seller} : ch_1(\boldsymbol{\nu}\, s).\ \mathsf{Seller} \to \mathsf{Shipper} : ch(\boldsymbol{\nu}\, s').\ \mathsf{Shipper} \to \mathsf{Depot} : ch_2(\boldsymbol{\nu}\, t)$$

Observe the second description is directly realisable as end-point processes, while the first one is not. Even if one may informally write down the first description, it is the second one which can have a precise correspondence with end-point behaviour. Thus we preclude descriptions like the first one, by demanding each participant acts only as a result of its local event. We call this principle *connectedness*. Connectedness is simply defined by tracking active/passive participants of each action, as formally given in [11]. Informally, for each A, A's sending action or its self-contained action (e.g. assignment and evaluation of a conditional guard) should always be immediately preceded by A's receiving action or its another self-contained action. Connectedness is closed under reductions.

Well-threadedness. The next condition is also about causality, but a slightly more subtle one. Consider the following connected interaction:

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

We claim that this global code (regardless of I) is unrealisable at end-points. In fact, the first action tells us that there is a thread in Buyer which invokes Seller. This thread becomes inactive in the second line where a service at ch_3 in Buyer is invoked. In the final line, Buyer communicates to Seller via s opened in the initial action. Written in the end-point calculus:

$$\begin{array}{rl}\mathsf{Buyer}[& \overline{ch_1}(\boldsymbol{\nu} s).\ \overline{s} \lhd \mathsf{op}\langle v\rangle.\ P \quad | \quad !\, ch_3(t).\ Q \quad]_{\sigma_1} \quad |\\ \mathsf{Seller}[& !\, ch_1(s).\ \overline{ch_2}(\boldsymbol{\nu} t).\ s \rhd \mathsf{op}(x).\ Q' \quad]_{\sigma_2} \quad |\\ \mathsf{Shipper}[& !\, ch_2(t).\ \overline{ch_3}(\boldsymbol{\nu} u).\ R \quad]_{\sigma_2}\end{array}$$

The first process of Buyer invokes ch_1 and sends v with operation op in the same session, while the second is a service at ch_3 (by **SCP** this channel should be ready to receive invocations). $\overline{s} \lhd \mathsf{op}\langle v\rangle$ cannot be located under ch_3, as it belongs to a session s. When the three processes interact, 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 other component*, the service at ch_3. Thus the sequencing in the global description gets violated.

The fundamental issue in the example above is that the given global code assumes a false, or 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 a 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*. For the formal definition, we first annotate a global interaction with identifiers for threads. *Annotated interactions*, denoted by $\mathcal{A}, \mathcal{A}', \ldots$, are given by the following grammar.

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

where $\tau_i \in \mathbb{N}$ (called *thread*) and $\tau_1 \neq \tau_2$ in the first two lines. Our task is to find a notion of "consistent annotation" so that causality specified globally is precisely realisable locally. We demand: if an input is annotated by τ then its directly succeeding output is annotated by τ again, similarly for self-contained actions; that two actions by $\mathcal{A}$ in the same session are annotated by the same thread; and that the input of session initiation is always given a fresh thread. We say I is *well-threaded* when it is connected and has a consistent annotation. If I is well-threaded and has no free session channels, it has a primary annotation from which all of its consistent annotations are derivable [11, §14, Prop. 11]. As an example, consider the following annotated interaction.

$$\begin{array}{l} \mathsf{Buyer}^{\tau_1} \to \mathsf{Seller}^{\tau_2} : ch_1(\boldsymbol{\nu}\, s).\ \mathsf{Seller}^{\tau_3} \to \mathsf{Shipper}^{\tau_4} : ch_2(\boldsymbol{\nu}\, t). \\ \quad \mathsf{Shipper}^{\tau_5} \to \mathsf{Buyer}^{\tau_6} : ch_3(\boldsymbol{\nu}\, u).\ \mathsf{Buyer}^{\tau_7} \to \mathsf{Seller}^{\tau_8} : s\langle \mathsf{op},\ v,\ x\rangle.\ I \end{array}$$

By the first two conditions, we have $\tau_1 = \tau_7$ and $\tau_6 = \tau_7$, hence $\tau_6 = \tau_1$, which violates the third condition. So this is *not* well-threaded. But the following annotated interaction is well-threaded:

$$\begin{array}{l} \mathsf{Buyer}^{1} \to \mathsf{Seller}^{2} : ch_1(\boldsymbol{\nu}\, s).\ \ \mathsf{Seller}^{2} \to \mathsf{Buyer}^{3} : ch_2(\boldsymbol{\nu}\, t). \\ \quad \mathsf{Buyer}^{3} \to \mathsf{Seller}^{2} : t\langle \mathsf{op}_1,\ v_1,\ x\rangle.\ \mathsf{Seller}^{2} \to \mathsf{Buyer}^{1} : s\langle \mathsf{op}_2,\ v_2,\ y\rangle.\ \mathbf{0} \end{array}$$

and in fact gives rise to the following correct end-points.

$$\begin{array}{lll} \mathsf{Buyer}[& \overline{ch_1}(\boldsymbol{\nu} s).\ \overline{s} \lhd \mathsf{op}_2\langle v_2\rangle.\mathbf{0} \quad | \quad !\, ch_2(t).\ t \rhd \mathsf{op}_1(x).\ \mathbf{0} &] \quad | \\ \mathsf{Seller}[& !\, ch_1(s).\ \overline{ch_2}(\boldsymbol{\nu} t).\ \overline{t} \lhd \mathsf{op}_1\langle v_1\rangle.s \rhd \mathsf{op}(y).\ \mathbf{0} &] \end{array}$$

There is a type discipline accepting all and only well-threaded interactions, from which we can derive a sound and complete algorithm for checking well-threadedness and for calculating, if any, (primary) consistent annotations [11].

Coherence. The final principle concerns consistency of descriptions of a behaviour belonging to the same service. We first note that it is often necessary to *merge* threads to obtain the final end-point behaviour of a single service. Consider the parallel composition:

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

where $\mathsf{op}_1 \neq \mathsf{op}_2$. Above, Buyer invokes Seller's service at ch twice in parallel. Now consider constructing the code for this service at channel ch: we need to merge these two threads into one end-point behaviour. But the global description is contradictory, since in one invocation the service reacts with op_1, while in the other the service reacts with op_2. As can be observed from this example, in a global description, the description of the behaviour of a single end-point can be *scattered in different portions of the code*. Hence we need to guarantee, in EPP, that these scattered descriptions are mergeable. This mergeablity condition is called *coherence*. Let $\mathcal{A}$ be consistently annotated. We list the key rules defining the partial operation $\mathsf{TP}(\mathcal{A}, \tau)$ (see [11] for a full definition):

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : b(\boldsymbol{\nu}\, \tilde{s}).\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \begin{cases} \overline{b}(\boldsymbol{\nu}\tilde{s}).\ \mathsf{TP}(\mathcal{A}, \tau_1) & \text{if } \tau = \tau_1 \\ !\, b(\tilde{s}).\ \mathsf{TP}(\mathcal{A}, \tau_2) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(A^{\tau_1} \to B^{\tau_2} : s\langle \mathsf{op}_i,\ e_i,\ x_i\rangle.\ \mathcal{A}, \tau) \stackrel{\text{def}}{=} \begin{cases} \overline{s} \lhd \mathsf{op}\langle e\rangle.\ \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_1 \\ s \rhd \mathsf{op}_i(x_i).\ \mathsf{TP}(\mathcal{A}, \tau) & \text{if } \tau = \tau_2 \\ \mathsf{TP}(\mathcal{A}, \tau) & \text{otherwise} \end{cases}$$

$$\mathsf{TP}(\mathcal{A}_1 +^{\tau'} \mathcal{A}_2, \tau) \stackrel{\text{def}}{=} \begin{cases} \mathsf{TP}(\mathcal{A}_1, \tau') \oplus \mathsf{TP}(\mathcal{A}_2, \tau') & \text{if } \tau = \tau' \\ \mathsf{TP}(\mathcal{A}_1, \tau) \sqcup \mathsf{TP}(\mathcal{A}_2, \tau) & \text{otherwise} \end{cases}$$

In the third rule, $\sqcup$ 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) $s \rhd \Sigma_{i\in J} \mathsf{op}_i(y_i).\ P_i \sqcup s \rhd \Sigma_{i\in K} \mathsf{op}_i(y_i).\ Q_i \stackrel{\text{def}}{=} \Sigma_{i\in J\cap K} \mathsf{op}_i(y_i).\ (P_i \sqcup Q_i) + \Sigma_{i\in J\setminus K} \mathsf{op}_i(y_i).\ P_i + \Sigma_{i\in K\setminus J} \mathsf{op}_i(y_i).\ Q_i$ with $P_i \bowtie Q_i$, where $P \bowtie Q$ says that the operation $P \sqcup Q$ is defined (thus we demand overlapping branches be mutually consistent); and (3) otherwise $P \sqcup Q$ is defined congruently up to $\equiv$. The partial operation $P \sqcup Q$ is called *merging operation*.

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. We say that $\mathcal{A}$ is *coherent* if it is consistently annotated (hence well-threaded) and $\mathsf{TP}(\mathcal{A}, \tau)$ is well-defined for each τ, and moreover satisfies: for each pair of threads τ_1, τ_2 in $\mathcal{A}$ such that $\tau_1 \equiv_{\mathcal{A}} \tau_2$, it holds that $\mathsf{TP}(\mathcal{A}, \tau_1) \sqcup \mathsf{TP}(\mathcal{A}, \tau_2)$ is defined. Coherence of a well-typed interaction is decidable [11, §15, Prop.13].

With coherence as the final principle, we can now project a well-structured global description to end-point processes that precisely realise the original global scenario (the projection is essentially invariant under different consistent annotations [11, §16.1, Prop.14]). Formally, 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 $(\boldsymbol{\nu}\tilde{s})\ \mathcal{A}$ under σ is defined as:

$$\mathsf{EPP}((\boldsymbol{\nu}\tilde{s})\ \mathcal{A}, \sigma) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\tilde{s})\ \Pi_{A\in\mathsf{part}(\mathcal{A})}\ A[\![\ \Pi_{[\tau]} \sqcup_{\tau'\in[\tau]} \mathsf{TP}(\mathcal{A}, \tau')\ ]\!]_\sigma$$

where ΠP_i denotes the parallel composition, $\mathsf{part}(\mathcal{A})$ denotes the set of participants mentioned in $\mathcal{A}$ and $[\tau]$ denotes the equivalence class $(\equiv_{\mathcal{A}})$ of τ.

4.2 Pruning and Main Theorem

Consider an interaction which is composed from two branches whose 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 obtain its EPP:

$$\mathsf{Buyer}[\overline{ch}(\boldsymbol{\nu} s).s \triangleright \mathsf{ack}.s \triangleleft \mathsf{go} \oplus \overline{ch}(\boldsymbol{\nu} s).s \triangleright \mathsf{ack}.s \triangleleft \mathsf{stop}] \mid \mathsf{Seller}[!\, ch(s).\overline{s} \triangleleft \mathsf{ack}.s \triangleright (\mathsf{go} + \mathsf{stop})]$$

Let us reduce the original global description, which, by dropping one branch, leads to $\mathsf{Seller} \to \mathsf{Buyer} : s\langle \mathsf{ack} \rangle$**.** $\mathsf{Buyer} \to \mathsf{Seller} : s\langle \mathsf{go} \rangle$. This EPP is:

$$\mathsf{Buyer}[\overline{ch}(\boldsymbol{\nu} s).s \triangleright \mathsf{ack}.s \triangleleft \mathsf{go}] \mid \mathsf{Seller}[!\, ch(s).\overline{s} \triangleleft \mathsf{ack}.s \triangleright \mathsf{go})]$$

Now we compare this end-point process with the reductum of the original EPP before, which is $\mathsf{Buyer}[\overline{ch}(\boldsymbol{\nu} s).s \triangleright \mathsf{ack}.s \triangleleft \mathsf{go}] \mid \mathsf{Seller}[!\, ch(s).\overline{s} \triangleleft \mathsf{ack}.s \triangleright (\mathsf{go} + \mathsf{stop})]$, where Seller has a redundant, useless branch "stop". This example shows that reduction in a global description can lose information which is still kept in the corresponding reduction in its EPP. This motivates the asymmetric relation of *pruning* $P \prec Q$, which indicates that if we cut off such unnecessary branches and replication from Q then we obtain P (see [11] for a formal definition). If $P \prec Q$, then P and Q are strong bisimilar under the minimal typing of P.

The main result of the paper follows. Below, by abuse of notation, I denotes consistently annotated interaction. $\equiv_\mu$ is the extension of $\equiv$ with the folding/unfolding of recursion. The proof is found in [11, §16.2–5]. (1) implies the lack of communication errors for the result of EPP.

Theorem 3 (end-point projection). *Let I be coherent, $\Gamma \vdash I \triangleright \Delta$ and $\Gamma \vdash \sigma$:*

(1) (type preservation) *If $\Gamma \vdash I \triangleright \emptyset$ and $\Gamma \vdash \sigma$, then $\Gamma \vdash \mathsf{EPP}(I, \sigma) \triangleright \emptyset$.*
(2) (soundness) *If $\mathsf{EPP}(I, \sigma) \to N$ then there exists (I', σ') such that $(\sigma,\ I) \to (\sigma',\ I')$ and $\mathsf{EPP}(I', \sigma') \prec \equiv_\mu N$.*
(3) (completeness) *If $(\sigma,\ I) \to (\sigma',\ I')$ then $\mathsf{EPP}(I, \sigma) \to N$ s.t. $\mathsf{EPP}(I', \sigma') \prec N$.*

5 Extensions and Future Work

Channel passing is a practically useful extension for business protocols, for example in the scenarios where participants need to send links to other participants. A typical example is when Buyer wants to buy from Seller, but Buyer does not know Seller's address (service channel) on the net. The only information Buyer has is a service channel of DirectoryService, which will send back the address of Seller to Buyer which in turn interacts with Seller through the obtained channel. Can we have a consistent EPP theory with unknown participants and channels? This has been an open problem left in WS-CDL's current specification (which allows

channel passing only for fixed participants). A possible extension of the EPP theory to channel passing, together with the treatment of other useful additional constructs, is discussed in [11]. Another interesting future topic is relaxations of the well-formedness principles while maintaining a sound EPP theory, on which some ideas are also discussed in [11].

The EPP theory has been developed with practical use in mind. There are several engineering scenes where the theory and its extensions may be useful.

- *Code generation.* We can create a complete distributed application by projecting a detailed global description to each of its end-points.
- *Prototype generation.* Projection can also be used for generating a *skeleton code* for each end-point which only contains basic communication behaviour, to be elaborated to full code. This is already used in [19].
- *Use of conformance.* A team of programmers initially agree on a shared global specification for communications among end-points: during/after programming, each programmer can check if her/his code conforms to the specification by conformance checking against projection. The conformance scheme is useful in other scenes, for example when we wish to check the usability of an existing service/library in a given global description.
- *Runtime monitoring, testing and debugging.* At runtime, each end-point can check if ongoing communications at his/her site conform to the global description by checking against its projection to that end-point. The monitoring can also be used for debugging and testing existing code.

Further, many static analyses/logical validation methods would become available for a global description from their well-developed end-point counterpart. The present work is intended as an initial trial towards a well-founded framework for communication-centred programming based on two distinct, and mutually complementary, descriptive paradigms, underpinned by a theory of EPP.

Acknowledgements. We thank Robin Milner for instigating and setting up the directions of our ongoing collaboration with W3C WS-CDL WG; the WG members, in particular Gary Brown, Steve Ross-Talbot and Nickolas Kavantzas for collaboration; and Joshua Guttman for his comments on an early version of the paper. This work is supported by EPSRC GR/T04236, GR/S55545, GR/S55538, GR/T04724, GR/T03208, GR/T03258 and IST2005-015905 MOBIUS.

References

1. Conversation with Steve Ross-Talbot. *ACM Queue*, 4(2), 2006.
2. Jos Baeten, Harm van Beek, and Sjouke Mauw. Specifying internet applications with DiCons. In *SAC'01*, pages 576–584. ACM Press, 2001.
3. Nick Benton, Luca Cardelli, and Cedric Fournet. Modern concurrency abstractions for C#. *ACM Trans. Program. Lang. Syst.*, 26(5):769–804, 2004.
4. Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In *TLCA'01*, volume 2044 of *LNCS*, pages 29–45, 2001.

5. K. Bhargavan, C. Fournet, and A. Gordon. Verified reference implementations of WS-Security protocols. In *WS-FN'06*, LNCS, 2006.
6. Eduardo Bonelli, Adriana B. Compagnoni, and Elsa L. Gunter. Correspondence assertions for process synchronization in concurrent communications. *JFP*, 15(2):219–247, 2005.
7. Sébastien Briais and Uwe Nestmann. A formal semantics for protocol narrations. In *TGC*, volume 3705, pages 163–181, 2005.
8. Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *Coordination*, volume 4038 of *LNCS*, pages 63–81, 2006.
9. M. Carbone, K. Honda, and N. Yoshida. A calculus of global interaction based on session types. In *DCM '06*, ENTCS, 2006.
10. M. Carbone, M. Nielsen, and V. Sassone. A calculus for trust management. In *FSTTCS '04*, volume 3328 of *LNCS*, pages 161–173. Springer, 2004.
11. Marco Carbone, Kohei Honda, Nobuko Yoshida, Robin Milner, Gary Brown, and Steve Ross-Talbot. A theoretical basis of communication-centred concurrent programming. To be published by W3C. Available at www.dcs.qmul.ac.uk/∼carbonem/cdlpaper, 2006.
12. Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida, and Sophia Drossopoulou. Session Types for Object-Oriented Languages. In *ECOOP'06*, volume 4067 of *LNCS*, pages 328–352, 2006.
13. Simon Gay and Malco Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, November 2005.
14. Claudio Guidi, Roberto Lucchi, Gianluigi Zavattaro, Nadia Busi, and Roberto Gorrieri. SOCK: a calculus for service oriented computing. In *ICSOC'06*, volume 4294 of *LNCS*, 2006.
15. J. D. Guttman, F. J. Thayer, and L. D. Zuck. The faithfulness of abstract protocol analysis: message authentication. In *CCS '01*, pages 186–195. ACM Press, 2001.
16. Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138, 1998.
17. Cosimo Laneve and Luca Padovani. Smooth orchestrators. In *FoSSaCS'06*, volume 3921 of *LNCS*, pages 32–46, 2006.
18. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.
19. PI4SOA. http://www.pi4soa.org.
20. Benjamin C. Pierce and David 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.
21. S. Ross-Talbot and T. Fletcher. WS-CDL Primer. To be published by W3C, 2006.
22. D. Sangiorgi. The name discipline of uniform receptiveness. In *ICALP'97*, volume 1256 of *LNCS*, pages 303–313. Springer, 1997.
23. Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *PARLE'94*, volume 817 of *LNCS*, pages 398–413, 1994.
24. W.M.P. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *Info. Tech. and Management*, 2(3):195–231, 2002.
25. Vasco Vasconcelos, António Ravara, and Simon J. Gay. Session types for functional multithreading. In *CONCUR'04*, volume 3170 of *LNCS*, pages 497–511, 2004.
26. W3C WS-CDL Working Group. Web services choreography description language version 1.0. http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/.