Domain-Aware Session Types

Luís Caires
Universidade Nova de Lisboa

Jorge A. Pérez
University of Groningen

Frank Pfenning
Carnegie Mellon University

Bernardo Toninho
Universidade Nova de Lisboa

Abstract

We develop a generalization of existing Curry-Howard interpretations of (binary) session types by relying on an extension of linear logic with features from hybrid logic, in particular modal worlds that indicate domains. These worlds govern domain migration, subject to a parametric accessibility relation familiar from the Kripke semantics of modal logic. The result is an expressive new typed process framework for domain-aware, message-passing concurrency. Its logical foundations ensure that well-typed processes enjoy session fidelity, global progress, and termination. Typing also ensures that processes only communicate with accessible domains and so respect the accessibility relation.

Remarkably, our domain-aware framework can specify scenarios in which domain information is available only at runtime; flexible accessibility relations can be cleanly defined and statically enforced. As a specific application, we introduce domain-aware multiparty session types, in which global protocols can express arbitrarily nested sub-protocols via domain migration. We develop a precise analysis of these multiparty protocols by reduction to our binary domain-aware framework: complex domain-aware protocols can be reasoned about at the right level of abstraction, ensuring also the principled transfer of key correctness properties from the binary to the multiparty setting.

1 Introduction

The goal of this paper is to show how existing Curry-Howard interpretations of session types [6, 7] can be generalized to a domain-aware setting by relying on an extension of linear logic with features from hybrid logic [35, 2]. As we discuss next, these extended logical foundations of message-passing concurrency allow us to analyze complex domain-aware concurrent systems (including those governed by multiparty protocols) in a precise and principled manner.

Software systems typically rely on communication between heterogeneous services; at their heart, these systems rely on message-passing protocols that combine mobility, concurrency, and distribution. As distributed services are often virtualized, protocols should span diverse software and hardware domains. These domains can have multiple interpretations, such as the location where services reside, or the principals on whose behalf they act. Concurrent behavior is then increasingly domain-aware: a partner’s potential for interaction is influenced not only by the domains it is involved in at various protocol phases (its context), but also by connectedness relations among domains. Moreover, domain architectures are rarely fully specified: to aid modularity and platform independence, system participants (e.g., developers, platform vendors, service clients) often have only partial views of actual domain structures.
Despite their importance in communication correctness and trustworthiness at large, the
formal status of domains within typed models of message-passing systems remains unexplored.

This paper contributes to typed approaches to the analysis of domain-aware commu-
nications, with a focus on session-based concurrency. This approach specifies the intended
message-passing protocols as session types [23, 24, 19]. Different type theories for binary
and multiparty (n-ary) protocols have been developed. In both cases, typed specifications
can be conveniently coupled with π-calculus processes [30], in which so-called session chan-
nels connect exactly two subsystems. Communication correctness usually results from two
properties: session fidelity (type preservation) and deadlock freedom (progress). The former
says that well-typed processes always evolve to well-typed processes (a safety property); the
latter says that well-typed processes will never get into a stuck state (a liveness property).

A key motivation for this paper is the sharp contrast between (a) the growing relevance
of domain-awareness in message-passing, concurrent systems and (b) the expressiveness of
existing session type frameworks, binary and multiparty, which cannot adequately specify
(let alone enforce) domain-related requirements. Indeed, existing session types frameworks,
including those based on Curry-Howard interpretations [6, 43, 10], capture communication
behavior at a level of abstraction in which even basic domain-aware assertions (e.g., “Shipper
resides in domain AmazonUS”) cannot be expressed. As an unfortunate consequence, the
effectiveness of the analysis techniques derived from these frameworks is rather limited.

To better illustrate our point, consider a common distributed design pattern: a middleware
agent (mw) which answers requests from clients (cl), sometimes offloading the requests to a
server (serv) to better manage local resource availability. In the framework of multiparty
session types [25] this can be represented as:

\[
\text{cl} \rightarrow \text{mw}: \{\text{request(req)}\}, \text{mw} \rightarrow \text{cl}: \{\text{reply(ans)}\}, \text{mw} \rightarrow \text{serv}: \{\text{done.end}\}, \text{wait} \rightarrow \text{serv}: \{\text{req(data)}\}, \text{serv} \rightarrow \text{mw}: \{\text{reply(ans)}\}, \text{mw} \rightarrow \text{cl}: \{\text{reply(ans).end}\}\}
\]

The client first sends a request to the middleware, which answers back with either a reply
message containing the answer or a wait message, signaling that the server will be contacted to
produce the final reply. While this multiparty protocol captures the intended communication
behavior, it does not capture that protocols for the middleware and the server often involve
some form of privilege escalation or specific authentication—ensuring, e.g., that the server
interaction is adequately isolated from the client, or that the escalation must precede the
server interactions. These requirements simply cannot be represented in existing frameworks.

Our work addresses this crucial limitation by generalizing Curry-Howard interpretations
of session types by appealing to hybrid logic features. We develop a logically motivated
typed process framework in which worlds from modal logics precisely and uniformly define
the notion of domain in session-based concurrency. At the level of binary sessions, domains
manifest themselves through point-to-point domain migration and communication. In
multiparty sessions, domain migration is specified choreographically through the new construct
\text{p moves \tilde{q} to \omega for } G_1; G_2, where participant \text{p} leads a migration of participants \text{q} to domain
(world) \omega in order to perform protocol \text{G}_1, who then migrate back to perform protocol \text{G}_2.

Returning to our example, let \text{Offload} ≜ \text{mw} \rightarrow \text{serv}: \{\text{req(data)}\}, \text{serv} \rightarrow \text{mw}: \{\text{reply(ans).end}\}. Our framework allows us to refactor the above protocol as:

\[
\text{cl} \rightarrow \text{mw}: \{\text{request(req)}\}, \text{mw} \rightarrow \text{cl}: \{\text{reply(ans)}\}, \text{mw} \rightarrow \text{serv}: \{\text{done.end}\}, \text{wait} \rightarrow \text{serv}: \{\text{init}\}, \text{mw} \rightarrow \text{serv to \omega_{priv} for \text{Offload}}; \text{mw} \rightarrow \text{cl}: \{\text{reply(ans).end}\}\}
\]

By considering a first-class multiparty domain migration primitive at the type and process
levels, we can specify that the offload portion of the protocol takes place after the middleware
and the server migrate to a private domain \omega_{priv}, as well as ensuring that only reachable
domains can be interacted with. For instance, the type for the server that is mechanically
projected from the protocol above ensures that the server first migrates to the private domain,
communicates with the middleware, and then migrates back to its initial domain.

Perhaps surprisingly, our domain-aware multiparty sessions are studied within a context
of logical binary domain-aware sessions, arising from a propositions-as-types interpretation
of hybrid linear logic [17, 13], with strong static correctness guarantees derived from the
logical nature of the system. Multiparty domain-awareness arises through an interpretation
of multiparty protocols as medium processes [4] that orchestrate the multiparty interaction
while enforcing the necessary domain-level constraints and migration steps.

Contributions The key contributions of this work are:
1. A process model with explicit domain-based migration (§2). We present a session
\(\pi\)-calculus with domains that can be communicated via novel domain movement prefixes.
2. A logic-based session type discipline for domain-aware interacting processes (§3). Building
upon an extension of linear logic with features from hybrid logic [17, 13] we generalize
the Curry-Howard interpretation of session types [6], by interpreting (modal) worlds as
domains where session behavior resides. Types include operators that specify domain
migration and communication. In our system, judgments stipulate the services that
processes implement and the domains where sessions should be present; domain mobility
is governed by a parametric accessibility relation. Our type discipline statically enforces
session fidelity, global progress and, notably, that communication can only happen between
reachable domains, thus introducing a new layer of reasoning to session type systems.
3. A systematic study of domain-aware multiparty sessions (§4), extending the standard
multiparty session framework of [25] with domain-aware migration and communication
primitives, at both the global orchestration and local typing levels. As a specific applica-
tion, our development leverages our logically motivated domain-aware binary sessions
to give a precise semantics to multiparty sessions through a notion of a typed medium
process that acts as an orchestrator of the domain-aware multiparty interaction, inheriting
the strong correctness properties of typed processes but now in a multiparty setting.
Crucially, we show that mediums soundly and completely encode the local behaviors of
participants in a domain-aware multiparty session.

We conclude with a discussion of related work (§5) and concluding remarks (§6). Ap-
pendix A lists omitted definitions and proofs. We point the interested reader to Appendices B
and C for extended examples on domain-aware multiparty and binary sessions, respectively.

2 Process Model

We introduce a variant of the synchronous \(\pi\)-calculus [37] with labeled choice and explicit
domain migration and communication. We write \(\omega, \omega', \omega''\) to stand for a concrete domain
\((w, w', \ldots)\) or a domain variable \((\alpha, \alpha', \ldots)\). Domains are handled at a high-level of ab-
straction, with their identities being attached to session channels. Just as the \(\pi\)-calculus
allows for communication over names and name mobility, our model also allows for domain
communication and mobility. These features are justified with the typing discipline of §3.

Definition 2.1. Given infinite, disjoint sets \(\Lambda\) of names \((x, y, z, u, v)\), \(L\) of labels \(l_1, l_2, \ldots,\)
\(W\) of domain tags \((w, w', w'')\) and \(V\) of domain variables \((\alpha, \beta, \gamma)\), respectively, the set of
processes \((P, Q, R)\) is defined by
\[
P ::= \quad 0 \quad | \quad P \mid Q \quad | \quad [x \leftrightarrow y] \quad | \quad x \mapsto \{k : P\} \quad | \quad (\nu y)P \quad | \quad x(y).P \quad | \quad x(y).P \quad | \quad x(y).P \quad | \quad x(y).P
\]
\[
\| x(y\otimes \omega).P \quad | \quad x(y\otimes \omega).P \quad | \quad x(\omega).P \quad | \quad x(\omega).P
\]
Domain-aware prefixes are present only in the last line. As we make precise in the typed setting of §3, these constructs realize mobility and domain communication, in the usual sense of the π-calculus: migration to a domain is always associated with mobility with a fresh name.

The operators 0 (inaction), \( P | Q \) (parallel composition) and \((\nu y)P\) (name restriction) are standard. We then have \( x(y).P \) (send \( y \) on \( x \) and proceed as \( P \)), \( x(y).P \) (receive \( z \) on \( x \) and proceed as \( P \) with parameter \( y \) replaced by \( z \)), and \( !x(y).P \) which denotes replicated (persistent) input. The forwarding construct \([x \leftrightarrow y]\) equates \( x \) and \( y \); it is a primitive representation of a copycat process. The last two constructs in the second line define a labeled choice mechanism: \( x \triangleright \{l_i : P_i\}_{i \in I} \) is a process that awaits some label \( l_j \) (with \( j \in I \)) and proceeds as \( P_j \). Dually, the process \( x \triangleleft l_i \) emits a label \( l_i \) and proceeds as \( P \).

The first two operators in the third line define explicit domain migration: given a domain \( \omega \), \( x(y@\omega).P \) denotes a process that is prepared to migrate the communication actions in \( P \) on endpoint \( x \), to session \( y \) on \( \omega \). Complementarily, process \( x(y@\omega).P \) signals an endpoint \( x \) to move to \( \omega \), providing \( P \) with the appropriate session endpoint that is then bound to \( y \). In a typed setting, domain movement will be always associated with a fresh session channel.

Alternatively, this form of coordinated migration can be read as an explicit form of agreement (or authentication) in trusted domains. Finally, the last two operators in the third line define output and input of domains, \( x(\omega).P \) and \( x(\alpha).P \), respectively. These constructs allow for domain information to be obtained and propagated across processes dynamically.

Following [36], we abbreviate \((\nu y)x(y)\) and \((\nu y)x(y@\omega)\) as \( \pi(y) \) and \( \pi(y@\omega) \), respectively. In \((\nu y)P\), \( x(y).P \), and \( x(y@\omega).P \) the distinguished occurrence of name \( y \) is binding with scope \( P \). Similarly for \( \alpha \) in \( x(\alpha).P \). We identify processes up to consistent renaming of bound names and variables, writing \( \equiv_\alpha \) for this congruence. \( P\{x/y\} \) denotes the capture-avoiding substitution of \( x \) for \( y \) in \( P \). While structural congruence \( \equiv \) expresses standard identities on the basic structure of processes (cf. Appendix A.1), reduction expresses their behavior.

Reduction \( (P \rightarrow Q) \) is the binary relation defined by the rules below; it specifies the computations that a process performs on its own.

\[
\begin{align*}
\text{x(y).Q | x(z).P} & \rightarrow Q | P\{y/z\} & \text{x(y).Q | !x(z).P} & \rightarrow Q | P\{y/z\} | !x(z).P \\
x(y@\omega).P | x(z@\omega).P & \rightarrow P | Q\{y/z\} & x(\omega).P | x(\alpha).Q & \rightarrow P | Q\{\omega/\alpha\} \\
(\nu x)([x \leftrightarrow y]) P & \rightarrow P\{y/x\} & Q \rightarrow Q' & \Rightarrow P \rightarrow Q \rightarrow P | Q' \\
P \rightarrow Q \Rightarrow (\nu y)P & \rightarrow (\nu y)Q & P \equiv P' \land P' \rightarrow Q' \land Q' \equiv Q \Rightarrow P \rightarrow Q \\
x \triangleright l_j : P | x \triangleright \{l_i : Q_i\}_{i \in I} & \rightarrow P | Q_j \quad (j \in I)
\end{align*}
\]

For the sake of generality, reduction allows dual endpoints with the same name to interact, independently of the domains of their subjects. The type system introduced next will ensure, among other things, local reductions, disallowing synchronisations among distinct domains.

## 3 Domain-aware Session Types via Hybrid Logic

This section develops a new domain-aware formulation of binary session types. Our system is based on a Curry-Howard interpretation of a linear variant of so-called hybrid logic, and can be seen as an extension of the interpretation of [6] to hybrid (linear) logic. Hybrid logic is often used as an umbrella term for a class of logics that extend the expressiveness of propositional logic by considering modal worlds as syntactic objects that occur in propositions.

As in [6, 7], propositions are interpreted as session types of communication channels, proofs as process typing derivations, and proof reduction as process communication. The main novelties of our approach are that we interpret worlds as domains, the hybrid connective @\( w \) \( A \) as the type of a session that migrates to an accessible domain \( w \), and type-level
quantification over worlds $\forall \alpha.A$ and $\exists \alpha.A$ as world communication. We also consider a type-level operator $\downarrow \alpha.A$ (read “here”) which binds the current domain of the session to $\alpha$ in $A$. The syntax of domain-aware session types is given in Def. 3.1, where $w, w_1, \ldots$ stand for worlds drawn from $W$, and where $\alpha, \beta$ and $\omega, \omega'$ are used as in the syntax of processes.

**Definition 3.1** (Domain-aware Session Types). The syntax of types $(A, B, C)$ is defined by

$$
A ::= 1 \mid A \rightarrow B \mid A \otimes B \mid \& \{k : A_i\}_{i \in t} \mid \oplus \{k : A_i\}_{i \in t} \mid !A
$$

Types are the propositions of intuitionistic linear logic where the additives $A \& B$ and $A \oplus B$ are generalized to a labelled $n$-ary variant. Propositions take the standard interpretation as session types, extended with hybrid logic operators [2], where modal worlds are subject to an accessibility relation that is tracked by a separate context $\Omega$. Intuitively, $\Omega$ collects direct reachability hypotheses of the form $\omega_1 \prec \omega_2$, meaning that domain $\omega_2$ is reachable from $\omega_1$. In our setting, types are assigned to channel names; a type assignment $x : A[\omega]$ enforces the use of name $x$ according to session $A$, in the domain $\omega$. A type environment is a collection of type assignments. Besides the accessibility context $\Omega$, our typing judgments consider two kinds of type environments, subject to different structural properties: a linear part $\Delta$ and an unrestricted part $\Gamma$. Our typing system is made up of two judgments:

(i) $\Omega \vdash \omega_1 \prec \omega_2$ and (ii) $\Omega; \Gamma; \Delta \vdash P :: z : A[\omega]$

Judgment (i) states that $\omega_1$ can directly reach $\omega_2$ under the hypotheses in $\Omega$. We omit $\Omega$ when it is clear from the reflexive, transitive closure of $\prec$, and $\omega_1 \nprec \omega_2$ when $\omega_1 \prec^{*} \omega_2$ does not hold. Judgment (ii) states that process $P$ offers the session behavior specified by type $A$ on channel $z$, with the session being localized at domain $\omega$, under the reachability hypotheses $\Omega$, using unrestricted sessions in $\Gamma$, and linear sessions in $\Delta$, where weakening and contraction principles hold for $\Gamma$ but not $\Delta$. Note that each hypothesis in $\Gamma$ and $\Delta$ is labeled with a specific domain. Empty contexts are written as ‘$\cdot$’.

**Typing Rules** Most typing rules are given in Fig. 1; the rest are listed in Appendix A.3.

Right rules (marked with $R$) specify how to offer a session of a given type, left rules (marked with $L$) define how to use a session. The hybrid nature of the system induces a notion of well-formedness of sequents: we consider a sequent $\Omega; \Gamma; \Delta \vdash P :: z : C[\omega_1]$ well-formed if $\Omega \vdash \omega_1 \prec^{*} \omega_2$ for every $x : A[\omega_2] \in \Delta$, which we abbreviate as $\Omega \vdash \omega_1 \prec^{*} \Delta$, meaning that all worlds mentioned in $\Delta$ are reachable from $\omega_1$ (not necessarily in a single direct step). No such world requirement is imposed on $\Gamma$. If an end sequent is well-formed, every sequent in its proof will also be well-formed. All rules (read bottom-up) preserve this invariant; only (cut), (copy), ($\otimes R$), ($\forall L$) and ($\exists R$) require explicit checks, which we discuss below. This invariant statically excludes interaction between sessions in unreachable domains (cf. Theorem 3.7).

We briefly discuss some of the typing rules, first noting that we always consider processes modulo structural congruence; hence, typability is closed under $\equiv$ by definition. Type $A \rightarrow B$ denotes a session that inputs a session of type $A$ and proceeds as $B$. To offer $z : A \rightarrow B$ at domain $\omega$, we input $y$ along $z$ that will offer $A$ at $\omega$ and proceed, now offering $z : B$ at $\omega$:

$$
\frac{\Omega; \Gamma; \Delta \vdash y : A[\omega] \vdash P :: z : B[\omega]}{(\rightarrow R) \quad \Omega; \Gamma; \Delta \vdash z(y) : P :: z : A \rightarrow B[\omega]}
$$

Dually, $A \otimes B$ denotes a session that outputs a session that will offer $A$ and continue as $B$. To offer $z : A \otimes B$, we perform an output along $z$ of a fresh name $y$, a session of type $A$ provided by $P$, and proceed as $Q$, offering $z : B$:

$$
\frac{\Omega; \Gamma; \Delta_1 \vdash P :: y : A[\omega] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z : B[\omega]}{(\otimes R) \quad \Omega; \Gamma; \Delta_1, \Delta_2 \vdash z(y) : (P | Q) :: z : A \otimes B[\omega]}
$$
The (cut) rule allows us to compose process $P$ offering $x:A[\omega_2]$ with $Q$ using $x:A[\omega_2]$ to offer $z:C[\omega_1]$, provided that world $\omega_2$ is reachable from $\omega_1$ (i.e., $\omega_1 \prec \omega_2$). Composition binds the name $x$. Note that we require domains in $\Delta_1$, the ambient sessions of the first premise, to be accessible from $\omega_1$, the domain of the second premise, which follows from the use of the transitive closure of accessibility $\prec$, using the intermediary $\omega_2$:

$$
\frac{
\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega \vdash \omega_1 \prec \Delta_1 \\
\Omega; \Gamma; \Delta_1 \vdash P :: x:A[\omega_2] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z:C[\omega_1]
}{
\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\nu x)(P | Q) :: z:C[\omega_1]
}
$$

Type 1 means that no further interaction will take place on the session; names of type 1 may be passed around as opaque values. $\{l_i : A_i\}_{i \in I}$ types a session channel that offers its partner a choice between the $A_i$ behaviors, each uniquely identified by a label $l_i$. Dually, $\exists\{l_i : A_i\}_{i \in I}$ types a session that selects some behavior $A_i$ by emitting the corresponding label. Type $\exists A$ types a shared (non-linear) channel, to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type $A$.

Following our previous remark on well-formed sequents, the only rules that appeal to accessibility are ($@R$), ($@L$), (copy), and (cut). These conditions are directly associated with varying degrees of flexibility in terms of typability, depending on what relationship is imposed between the world to the left and to the right of the turnstile in the left rules. Notably, our system leverages the accessibility judgment to enforce that communication is only allowed between processes whose sessions are in (transitively) reachable domains.

The type operator $@\omega$ realizes a domain migration mechanism which is specified both at the level of types and of processes via name mobility tagged with a domain name. Thus, a channel typed with $@\omega A$ denotes that behavior $A$ is available by first moving to domain $\omega_2$, directly accessible from the current domain. More precisely, we have:

$$
\frac{
\Omega \vdash \omega_1 \prec \omega_2 \\
\Omega; \Gamma; \Delta \vdash P :: y:A[\omega_2]
}{
\Omega; \Gamma; \Delta \vdash z@\omega_2 A[\omega_1]
}
$$

Hence, a process offering a behavior $z:@\omega_2 A$ at $\omega_1$ ensures: (i) behavior $A$ is available at $\omega_2$ along a fresh session channel $y$ that is emitted along $z$ and (ii) $\omega_2$ is directly reachable from $\omega_1$. To maintain well-formedness of the sequent we also must check that all domains in $\Delta$ are still accessible from $\omega_2$. Dually, using a service $x:@\omega_2 A[\omega_2]$ entails receiving a channel $y$ that will offer behavior $A$ at domain $\omega_3$ (and also allowing the usage of the fact that $\omega_2 \prec \omega_3$).

Domain-quantified sessions introduce domains as fresh parameters to types: a particular service can be specified with the ability to refer to any existing directly reachable domain (via universal quantification) or to some a priori unspecified reachable domain:

$$
\frac{
\Omega, \omega_1 \prec \alpha; \Gamma; \Delta \vdash P :: z:A[\omega_1] \\
\alpha \notin \Omega, \Delta, \omega_1
}{
\Omega, \Gamma; \Delta \vdash z(\alpha).P :: z:\forall \alpha.A[\omega_1]
}
$$

Rule ($\forall R$) states that a process seeking to offer $\forall \alpha.A[\omega_1]$ denotes a service that is located at domain $\omega_1$ but that may refer to any fresh domain directly reachable from $\omega_1$ in its specification (e.g. through the use of $@$). Operationally, this means that the process must be ready to receive from its client a reference to the domain being referred to in the type, which is bound to $\alpha$ (occurring fresh in the typing derivation). Dually, Rule ($\forall L$) indicates that a process interacting with a service of type $x:\forall \alpha.A[\omega_2]$ must make concrete the domain that is directly reachable from $\omega_2$ it wishes to use, which is achieved by the appropriate output action. Rules ($\exists L$) and ($\forall R$) for the existential quantifier have a dual reading.

Finally, we introduce a type-level operator $\downarrow \alpha.A$ which allows for a session type to refer to its current domain in a dynamic way:
The typing rules that govern $\downarrow\alpha.A$ are completely symmetric and produce no action at the process level, merely instantiating the world variable $\alpha$ with the current domain $\omega$ of the session. As will be made clear in §4, this connective plays a crucial role in ensuring the correctness of our analysis of multiparty domain-aware sessions in our logical setting.

By developing our type theory with a separate domain accessibility judgment, we can consider a particular accessibility relation as a parameter of the framework. This allows changing accessibility relations and its properties without having to alter the entire system. To consider the simplest possible accessibility relation, the only defining rule for accessibility would be Rule (\texttt{whyp}) in Fig. 1. To consider an accessibility relation which is an equivalence relation we would add reflexivity, transitivity, and symmetry rules to the judgment.

**Discussion and Examples** Crucially, our domain-aware theory is conservative with respect to the Curry-Howard interpretation of session types in [6, 7]: our type theory is a process interpretation of hybridized linear logic (whose rules can be recovered by erasing the process terms), of which [6, 7] is a special case where every session resides at the same domain.

Conversely, a fundamental consequence of our hybrid interpretation is that it refines the session type structure in non-trivial ways. By requiring that communication only happen between processes located at the same (or reachable) worlds we effectively introduce a new layer of reasoning to session type systems. To illustrate this feature, consider the following session type, specifying a simplified interaction between a web store and its clients:

\[
\text{WStore} \triangleq \text{addCart} \to \{\text{buy} : \text{Pay} \text{, quit} : 1\} \quad \text{Pay} \triangleq \text{CCNum} \to \oplus\{\text{ok} : \text{Rcpt} \odot 1 \text{, nok} : 1\}
\]

The web store specification, representable in existing session type systems (e.g. [6, 43, 24]), allows clients to checkout their shopping carts by emitting a \texttt{buy} message or to \texttt{quit}. In the former case, the client pays for the purchase by sending their credit card data. Despite describing the intended communication patterns correctly, the types above do not capture the crucial fact that in a realistic setting, the client’s sensitive information should only be requested after entering a secure domain. We can address this limitation, by using the type-level domain migration construct, refining the types \text{WStore} and \text{Pay} above as follows:

\[
\text{WStore}_{\text{sec}} \triangleq \text{addCart} \to \&\{\text{buy} : \text{@sec\ }\text{Pay}, \text{quit} : 1\} \quad \text{Pay}_{\text{sec}} \triangleq \text{CCNum} \to \oplus\{\text{ok} : \text{@sec\Rcpt} \odot 1 \text{, nok} : 1\}
\]

\text{WStore}_{\text{sec}} now decrees that communication behavior pertinent to type \text{Pay} should be preceded by a migration step to the trusted domain \text{sec}, which should be directly reachable from \text{WStore}_{\text{sec}}’s current domain. The type can also specify that the receipt must originate from a bank domain \text{bnk} (e.g., ensuring the client that it is not produced by the store without entering the bank domain). When considering the interactions with a client (at domain \text{c}) that checks out their cart, we reach a state that is typed with the judgment below (left side):

\[
\text{c} \prec \text{sec} ; ; x : \text{@sec\ }\text{Pay}[\text{ws}] \vdash \text{Client} :: z : \text{@sec\ }\text{1}[\text{c}] \quad \text{c} \prec \text{us} , \text{ws} \prec \text{sec} ; ; x' : \text{Pay}[\text{sec}] \vdash \text{Client'} :: z' : \text{1}[\text{sec}]
\]

At this point, it is impossible for a (typed) client to interact with the behavior that is protected by the domain \text{sec}, since it is not the case that \text{c} $\prec^* \text{sec}$. This ensures, e.g., that a client cannot exploit the payment platform of the web store by accessing the trusted domain in unforeseen ways. The client can only communicate in the secure domain after the web store service has migrated accordingly, as shown by the judgment above (right side).
Technical Results. We state the main results of type safety via type preservation (Theorem 3.3) and global progress (Theorem 3.4). These results directly ensure session fidelity and deadlock-freedom. Typing also ensures that termination, i.e., processes do not exhibit infinite reduction paths (Theorem 3.5). Moreover, as a property specific to domain-aware processes, we show domain preservation, i.e., processes respect their domain accessibility conditions (Theorem 3.7). The formal development of these results relies on a domain-aware labeled transition system (Appendix A.2). Its definition is a simple generalization of the early labelled transition system for the session π-calculus.

Type Safety and Termination. Following [6], our proof of type preservation relies on a simulation between reductions in the session-typed π-calculus and logical proof reductions.

Lemma 3.2 (Domain Substitution). If \( \Omega \vdash \omega_1 < \omega_2 \text{ then } \Omega, \omega_1 \alpha < \omega_2 \Omega, \alpha; \Gamma \vdash P : z : A[\alpha] \text{ then } \Omega, \alpha; \Gamma \vdash P[\omega_2/\alpha] : z : A[\alpha] \\{ \omega_2/\alpha \}] = A[\omega_2/\alpha].

Lemma 3.3 (Type Preservation). If \( \Omega; \Gamma \vdash P : z : A[\omega] \text{ and } P \rightarrow Q \text{ then } \Omega; \Gamma \vdash Q : z : A[\omega].

Proof (Sketch). The proof mirrors those of [6, 5, 39], relying on a series of lemmas (Appendix A.4) relating the result of dual process actions (via our LTS semantics) with typable parallel compositions through the (cut) rule. For session type constructors of [6], the results are unchanged. For the domain-aware session type constructors, the development is identical that of [5] and [39], which deal with communication of types and data terms, respectively.
The proof of global progress (following [6]) relies on a notion of a live process, which intuitively consists of a process that has not yet fully carried out its ascribed session behavior, and thus is a parallel composition of processes where at least one is a non-replicated process, guarded by some action. Formally, we define live(P) if and only if P ≡ (νn)(π.Q | R), for some R, names n and a non-replicated guarded process π.Q.

**Theorem 3.4** (Global Progress). If Ω; · ⊢ P :: x:1[ω] and live(P) then ∃Q s.t. P → Q.

Note that Theorem 3.4 is without loss of generality since using the cut rules we can compose arbitrary well-typed processes together and x need not occur in P due to Rule (1R).

Termination (strong normalization) is a relevant property for interactive systems: while from a global perspective they are meant to run forever, at a local level participants should always react within a finite amount of time, and never engage into infinite internal behavior. We say that a process P terminates, noted P ⊥, if there is no infinite reduction path from P.

**Theorem 3.5** (Termination). If Ω; Γ; ∆ ⊢ P :: x:A[ω] then P ⊥.

**Proof (Sketch).** By adapting the linear logical relations given in [34, 5]. For the system in §3 without quantifiers, the logical relations correspond to those in [34], extended to carry over Ω. When considering quantifiers, the logical relations resemble those proposed for polymorphic session types in [5], noting that no impredicativity concerns are involved. ▶

**Domain Preservation.** As a consequence of the hybrid nature of our system, well-typed processes are guaranteed not only to faithfully perform their prescribed behavior in a deadlock-free manner, but they also do so without breaking the constraints put in place on domain reachability given by our well-formedness constraint on sequents.

**Theorem 3.6.** Let E be a derivation of Ω; Γ; ∆ ⊢ P :: z:A[ω]. If Ω; Γ; ∆ ⊢ P :: z:A[ω] is well-formed then every sub-derivation in E well-formed.

While unreachable domains can appear in Γ, such channels can never be used and thus can not appear in a well-typed process due to the restriction on the (copy) rule. Combining Theorems 3.3 and 3.6 we can then show:

**Theorem 3.7.** Let (1) Ω; Γ; Δ, Δ’ ⊢ νx)(P | Q) :: z : A[ω], (2) Ω; Γ; Δ ⊢ P :: x:B[ω’], and (3) Ω; Γ; Δ’, x:B[ω’] ⊢ Q :: z:A[ω]. If (νx)(P | Q) → (νx)(P’ | Q’) then: (a) Ω; Γ; Δ ⊢ P’ :: x’.B’[ω’’], for some x’, B’, ω’’; (b) Ω; Γ; Δ’, Δ’ ⊢ P’ :: x’.B’[ω’’] → Q’ :: z:A[ω]; (c) ω ≡ ω’’.

Theorem 3.7 characterizes reachability wrt reduction, showing that even if an ambient session changes domains, typing ensures that such a domain will be (transitively) accessible.

## 4 Domain-Aware Multiparty Session Types

We now shift our attention to multiparty session types [25]. We consider the standard ingredients: global types, local types, and the projection function that connects the two. Our global types include a new domain-aware construct, p.moves q.to.ω for G₁; G₂; our local types exploit the hybrid session types from Def. 3.1. Rather than defining a separate type system based on local types for the process model of §2, our analysis of multiparty protocols extends the approach defined in [4], which uses medium processes to characterize correct multiparty implementations. The advantages are twofold: on the one hand, medium processes provide a precise semantics for global types; on the other hand, they enable the principled transfer of the correctness properties established in §3 for binary sessions (type preservation, global
progress, termination, domain preservation) to the multiparty setting. Below, *participants* are ranged over by \( p, q, r, \ldots \); we write \( \tilde{q} \) to denote a finite set of participants \( q_1, \ldots, q_n \).

Besides the new domain-aware global type, our syntax of global types includes constructs from [25, 16]. We consider value passing in branching (cf. \( U \) below), fully supporting delegation. To streamline the presentation, we consider global types without recursion.

**Definition 4.1 (Global and Local Types).** Define global types \( (G) \) and local types \( (T) \) as

\[
U :::= \text{bool} \mid \text{nat} \mid \text{str} \mid \ldots \mid T
\]

\[
G ::= \text{end} \mid p \rightarrow q \{l\{U_i\}, G_i\}_{i \in I} \mid p \text{ moves } \tilde{q} \text{ to } \omega \text{ for } G_1; G_2
\]

\[
T ::= \text{end} \mid p\{k\{U_i\}, T_i\}_{i \in I} \mid \text{p}!\{k\{U_i\}, T_i\}_{i \in I} \mid \forall \alpha.T \mid \exists \alpha.T \mid \emptyset \alpha.T \mid \downarrow \alpha.T
\]

The completed global type is denoted \text{end}. Given a finite \( I \) and pairwise different labels, \( p \rightarrow q \{l\{U_i\}, G_i\}_{i \in I} \) specifies that by choosing label \( l_i \), participant \( p \) may send a message of type \( U_i \) to participant \( q_i \), and then continue as \( G_i \). We decree \( p \neq q \), so reflexive interactions are disallowed. The global type \( p \text{ moves } \tilde{q} \text{ to } \omega \text{ for } G_1; G_2 \) specifies the migration of participants \( p, \tilde{q} \) to domain \( \omega \) in order to perform the sub-protocol \( G_1 \); this migration is lead by \( p \). Subsequently, all of \( p, \tilde{q} \) migrate from \( \omega \) back to their original domains and protocol \( G_2 \) is executed. This intuition will be made precise by the medium processes for global types (cf. Def. 4.8). Notice that \( G_1 \) and \( G_2 \) may involve different sets of participants. In writing \( p \text{ moves } \tilde{q} \text{ to } \omega \text{ for } G_1; G_2 \) we assume two natural conditions: (a) all migrating participants intervene in the sub-protocol (i.e., the set of participants of \( G_1 \) is exactly \( p, \tilde{q} \)) and (b) domain \( \omega \) is accessible (in the sense of \( \prec \)) by all these migrating participants in \( G_1 \).

**Definition 4.2.** The set of participants of \( G \) (denoted \( \text{part}(G) \)) is defined as: \( \text{part} (\text{end}) = \emptyset \), \( \text{part} (p \rightarrow q \{l\{U_i\}, G_i\}_{i \in I}) = \{p, q\} \cup \bigcup_{i \in I} \text{part}(G_i) \), \( \text{part}(p \text{ moves } \tilde{q} \text{ to } \omega \text{ for } G_1; G_2) = \{p\} \cup \tilde{q} \cup \text{part}(G_1) \cup \text{part}(G_2) \). We sometimes write \( p \in G \) to mean \( p \in \text{part}(G) \).

Global types are projected onto participants so as to obtain local types. The terminated local type is \text{end}. The local type \( p^?\{k\{U_i\}, T_i\}_{i \in I} \) denotes an offer of a set of labeled alternatives; the local type \( p^!\{k\{U_i\}, T_i\}_{i \in I} \) denotes a behavior that chooses one of such alternatives. Exploiting the domain-aware framework in §3, we introduce four new local types. They increase the expressiveness of standard local types by specifying universal and existential quantification over domains (\( \forall \alpha.T \) and \( \exists \alpha.T \)), migration to a specific domain (\( \emptyset \alpha.T \)), and a reference to the current domain (\( \downarrow \alpha.T \), with \( \alpha \) occurring in \( T \)).

We now define (merge-based) projection for global types [16]. To this end, we rely on a merge operator on local types, which in our case considers messages \( U \).

**Definition 4.3 (Merge).** We define \( \sqcup \) as the commutative partial operator on base and local types such that \( \text{bool} \sqcup \text{bool} = \text{bool} \) (and analogously for other base types), and

1. \( T \sqcup T = T \), where \( T \) is one of the following: \text{end}, \( \square\{k\{U_i\}, T_i\}_{i \in I}, \emptyset \omega.T, \forall \alpha.T, \exists \alpha.T; \)
2. \( p^?\{k\{U_k\}, T_k\}_{k \in K} \sqcup p^!\{l\{U'_i\}, T'_i\}_{i \in J} =
\]

\[
p^?\{k\{U_k\}, T_k\}_{k \in K} \cup \{l\{U'_i\}, T'_i\}_{i \in J} \cup \{k\{U_i \sqcup U'_i\}, (T_i \sqcup T'_i)\}_{i \in K \cup J}
\]

and is undefined otherwise.

Therefore, for \( U_1 \sqcup U_2 \) to be defined there are two options: (a) \( U_1 \) and \( U_2 \) are identical base, terminated, selection, or "hybrid" local types; (b) \( U_1 \) and \( U_2 \) are branching types, but not necessarily identical: they may offer different options but with the condition that the behavior in labels occurring in both \( U_1 \) and \( U_2 \) must be mergeable.

To define projection and medium processes for the new global type \( p \text{ moves } \tilde{q} \text{ to } \omega \text{ for } G_1; G_2 \), we require ways of "fusing" local types and processes. The intent is to capture in a single (sequential) specification the behavior of two distinct (sequential) specifications, i.e., those corresponding to protocols \( G_1 \) and \( G_2 \). For local types, we have the following definition:
When a side condition does not hold, the map is undefined. (WF, in the following)

The local type for \( \varnothing \) moves \( \langle \alpha.\{l\rangle \) to \( \langle \alpha.\{l\rangle \) and is undefined otherwise.

Definition 4.5 (Merge-based Projection [16]). Let \( G \) be a global type. The merge-based projection of \( G \) under participant \( r \), denoted \( G | r \), is defined as \( \text{end} | r = \text{end} \) and

\[
\begin{align*}
\text{p} \mapsto \text{q} &\colon \{l\} \{G\} \mid r = \\
\{l\} \{G\} \mid r &\mid \cup \{l\} \{G\} \mid r \text{ otherwise (\( \cup \) as in Def. 4.3)}
\end{align*}
\]

When a side condition does not hold, the map is undefined.

The projection for the type \( \text{p moves} \text{q} \) to \( \omega \) for \( G_1 ; G_2 \) is one of the key points in our analysis.

The local type for \( \text{p} \), the leader of the migration, starts by binding the identity of its current domain (say, \( \omega_p \)) to \( \beta \). Then, the (fresh) domain \( \omega \) is communicated, and there is a migration step to \( \omega \), where \( \omega \) is the protocol \( G_1 | \text{p} \) will be performed. Finally, there is a migration step from \( \omega \) back to \( \omega_p \); once there, the protocol \( G_2 | \text{p} \) will be performed. The local type for all of \( \text{q} \in \text{q} \) follows accordingly: they expect \( \omega \) from \( \text{p} \); the migration from their original domains to \( \omega \) (and back) is as for \( \text{p} \). For participants in \( G_1 \), the fusion on local types (Def. 4.4) defines a local type that includes the actions for \( G_1 \) but also for \( G_2 \), if any: a participant in \( G_1 \) need not be involved in \( G_2 \). Interestingly, the resulting local types \( \downarrow(\exists \alpha. \exists \beta G_1 | \text{p}) \circ @_\beta G_2 | \text{p} \) and \( \downarrow(\exists \alpha. \exists \beta G_1 | \text{q}) \circ @_\beta G_2 | \text{q} \) define a precise combination of hybrid connectives whereby each migration step is bound by a quantifier or the current domain.

The following notion of well-formedness for global types is standard:

Definition 4.6 (Well-Formed Global Types [25]). We say that global type \( G \) is well-formed (WF, in the following) if the projection \( G | \text{r} \) is defined for all \( \text{r} \in G \).

Analyzing Global Types via Medium Processes

A medium process is a well-typed process from §2 that captures the communication behavior of the domain-aware global types of Def. 4.1. Here we define medium processes and establish two fundamental characterization results for them (Theorems 4.11 and 4.12). We shall consider names indexed by participants: given a name \( c \) and a participant \( \text{p} \), we use \( c_\text{p} \) to denote the name along which the session behavior of \( \text{p} \) will be made available. This way, if \( \text{p} \neq \text{q} \) then \( c_\text{p} \neq c_\text{q} \).

To define mediums, we need to fuse sequential processes just as Def. 4.4 fuses local types:

Definition 4.7 (Fusion of Processes). We define \( \circ \) as the partial operator on well-typed processes such that (with \( \pi \in \{c(y), c(\omega), c(\alpha), c(y @ \omega), c(\alpha @ \omega), c(\alpha @ l)\} \) :

\[
c(y).\{[u @ y] \mid P \} \circ Q \triangleq c(y).\{[u @ y] \mid (P \circ Q)\} \\
c(\alpha).\{[l] \mid P_i\} \circ Q \triangleq c(\alpha).\{[l] \mid (P_i \circ Q)\} \quad (P \circ Q) \triangleq Q \\
(\pi P) \circ Q \triangleq (\pi \circ Q)
\]

and is undefined otherwise.

The previous definition suffices for the definition of medium processes, given next. Using indexed names, a medium process uniformly captures the behavior of a global type:
Definition 4.8 (Medium Process). Let $G$ be a global type (cf. Def. 4.1), $\bar{c}$ be a set of indexed names, and $\bar{\omega}$ a set of domains. The medium process of $G$ (or simply medium), denoted $M^\omega[G](\bar{c})$, is defined as follows:

$$M^\omega[G](\bar{c}) = \begin{cases} 0 & \text{if } G = \text{end} \\ c_p \triangleright \{ \lambda_1 : c_p(u).c_q(u) \triangleright c_{\bar{v}}(u).([u \leftrightarrow v] \mid M^\omega[G_1](\bar{c})) \}_{i \in I} & \text{if } G = p \Rightarrow q \{ l_k(U_i).G_i \}_{i \in I} \\ c_p(\alpha).c_q(\bar{\alpha}) & \text{if } G = p \text{ moves } q_1, \ldots, q_n \text{ to } w \text{ for } G_1 ; G_2 \\ c_p(y_0 \otimes \alpha)_0.c_q(y_1 \otimes \alpha)_1.c_q(\cdots).c_q(y_n \otimes \alpha)_n & \text{if } G = p \text{ moves } q_1, \ldots, q_n \text{ to } w \text{ for } G_1 ; G_2 \\ M^\omega((\bar{\omega}/\alpha_1/\alpha_2/\ldots/\alpha_n)\{G_1\}(\bar{\gamma})) \circ \{ \gamma_0(m_p \otimes \alpha_p).y_1(m_q_1 \otimes \alpha_q_1).\cdots.y_n(m_q_n \otimes \alpha_q_n).M^\omega[G_2](\bar{m}_n) \} & \text{if } G = p \text{ moves } q_1, \ldots, q_n \text{ to } w \text{ for } G_1 ; G_2 \\ \end{cases}$$

where $M^\omega[G_1](\bar{c}) \circ M^\omega[G_2](\bar{c})$ is as in Def. 4.7.

The medium for $G = p \Rightarrow q \{ l_k(U_i).G_i \}_{i \in I}$ exploits four prefixes to mediate in the interaction between the implementations of $p$ and $q$: the first two prefixes (on name $c_p$) capture the label selected by $p$ and the subsequently received value; the third and fourth prefixes (on name $c_q$) propagate the choice and forward the value sent by $p$ to $q$.

The medium for $G = p \text{ moves } q_1, \ldots, q_n \text{ to } w \text{ for } G_1 ; G_2$ showcases the expressivity and convenience of our domain-aware process framework. In this case, the medium’s behavior takes place through the following steps: First, $M^\omega[G]\bar{c}$ inputs a domain identifier (say, $\omega$) from $p$ which is forwarded to $q_1, \ldots, q_n$, the other participants of $G_1$. Secondly, the roles $p, q_1, \ldots, q_n$ migrate from their domains $\omega_1, \omega_2, \ldots, \omega_n$ to $\omega$. At this point, the medium for $G_1$ can execute, keeping track the current domain $\omega$ for all participants. Finally, the participants of $G_1$ migrate back to their original domains and the medium for $G_2$ executes.

Recalling the domain-aware global type of §1, we produce its medium process:

$$c_{c_1} \triangleright \{ \text{request } : c_{c_2}(r).c_{c_3} \triangleright \text{request; } c_{\bar{c}_2}(\bar{v}).([r \leftrightarrow \bar{v}] \mid \text{done } : c_{\bar{c}_3} \triangleright \text{done; } 0) \} \},$$
$$c_{c_2} \triangleright \{ \text{reply } : c_{\bar{c}_3}(a).c_{c_1} \triangleright \text{reply; } c_{\bar{c}_3}(n).([a \leftrightarrow \bar{n}] \mid \text{done } : c_{\bar{c}_3} \triangleright \text{done; } 0) \} \},$$
$$c_{c_1} \triangleright \{ \text{wait } : c_{\bar{c}_2}(\bar{n}).([a \leftrightarrow n] \mid \text{done } : c_{\bar{c}_3} \triangleright \text{done; } 0) \} \} \},$$

The medium ensures the client’s domain remains fixed through the entire interaction, regardless of whether the middleware chooses to interact with the server. This showcases how our medium transparently manages domain migration of participants.

Characterization Results We state results that offer a sound and complete account of the relationship between: (i) a global type $G$ (and its local types), (ii) its medium process $M^\omega[G](\bar{c})$, and (iii) process implementations for the participants $\{p_1, \ldots, p_n\}$ of $G$. In a nutshell, these results say that the typeful composition of $M^\omega[G](\bar{c})$ with processes for each $p_1, \ldots, p_n$ (well-typed in the system of §3) performs the intended global type. Crucially, these processes reside in distinct domains and can be independently developed, guided by their local type—they need not know about the medium’s existence or structure. The results generalize those in [4] to the domain-aware setting. Given a global type $G$ with $\text{part}(G) = \{ p_1, \ldots, p_n \}$, below we write $\text{npart}(G)$ to denote the set of indexed names $\{ c_{p_1}, \ldots, c_{p_n} \}$. We define:

Definition 4.9 (Compositional Typing). We say $\Omega; \Gamma; \Delta \vdash M^\omega[G](\bar{c}) :: z : C$ is a compositional typing if: (i) it is a valid typing derivation; (ii) $\text{npart}(G) \subseteq \text{dom}(\Delta)$; and (iii) $C = 1$. 
A compositional typing says that $M^2[G](\Check{c})$ depends on behaviors associated to each participant of $G$; it also specifies that $M^2[G](\Check{c})$ does not offer any behaviors of its own.

The following definition relates binary session types and local types: the main difference is that the former do not mention participants. Below, $B$ ranges over base types ($\text{bool, nat, \ldots}$).

**Definition 4.10 (Local Types → Binary Types).** Mapping $\langle\cdot\rangle$ from local types $T$ (Def. 4.1) into binary types $A$ (Def. 3.1) is inductively defined as $\langle B \rangle = 1$ and

\[
\begin{align*}
\langle p!\{k(U_i);T_i\}\rangle & = \{k : \langle U_i \rangle \otimes \langle T_i \rangle\}\rangle_{i \in I} & \langle \forall \alpha.T \rangle & = \forall \alpha.\langle T \rangle \\
\langle p?\{k(U_i);T_i\}\rangle & = \{k : \langle U_i \rangle \to \langle T_i \rangle\}\rangle_{i \in I} & \langle \exists \alpha.T \rangle & = \exists \alpha.\langle T \rangle \\
\langle @\omega.T \rangle & = \omega.\langle T \rangle & \langle \perp.\alpha.T \rangle & = \perp.\langle T \rangle
\end{align*}
\]

Our first characterization result ensures that well-formedness of a global type $G$ guarantees the typability of its medium $M^2[G](\Check{c})$ using binary session types. Hence, it ensures that multiparty protocols can be analyzed by composing the medium with independently obtained, well-typed implementations for each protocol participant. Crucially, the resulting well-typed process will inherit all correctness properties ensured by binary typability established in §3.

**Theorem 4.11 (Global Types → Typed Mediums).** If $G$ is WF with $\text{part}(G) = \{p_1, \ldots, p_n\}$ then $\Omega; \Gamma; c_{p_1} : \langle G[p_1]\rangle[\omega_1], \ldots, c_{p_n} : \langle G[p_n]\rangle[\omega_n] \vdash M^2[G](\Check{c}) : z : 1[\omega_m]$ is a compositional typing, for some $\Omega, \Gamma$, with $\omega = \omega_1, \ldots, \omega_n$. We assume that $\omega_i \prec \omega_n$ for all $i \in \{1, \ldots, n\}$ (the medium’s domain is accessible by all), and that $i \neq j$ implies $\omega_i \neq \omega_j$.

**Proof.** By induction on the structure of $G$; see Appendix A.6. ▲

The second characterization result, given next, is the converse of Theorem 4.11: binary typability precisely delineates the interactions that underlie well-formed multiparty protocols.

We need an auxiliary relation on local types, written $\propto^\triangleright_\omega$, that relates types with branching and “here” type operators, which have silent process interpretations (cf. Figure 1 and Appendix A.3). First, we have $T_1 \propto^\triangleright_\omega T_2$ if there is a $T'$ such that $T_1 \sqcup T = T_2$ (cf. Def. 4.3). Second, we have $T_1 \propto^\triangleright_\omega T_2$ if (i) $T_1 = T'$ and $T_2 = \perp.\alpha.T'$ and $\alpha$ does not occur in $T'$; but also if (ii) $T_2 = \perp.\alpha.T'$ and $T_2 = T'[\omega/\alpha]$ (see Appendix A.5 for a formal definition of $\propto^\triangleright_\omega$).

**Theorem 4.12 (Well-Typed Mediums → Global Types).** Let $G$ be a global type (cf. Def. 4.1). If $\Omega; \Gamma; c_{p_1} : A_1[\omega_1], \ldots, c_{p_n} : A_n[\omega_n] \vdash M^2[G](\Check{c}) : z : 1[\omega_m]$ is a compositional typing then $\exists T_1, \ldots, T_n$ such that $G[p_j] \propto^\triangleright_\omega T_j$ and $\langle T_j \rangle = A_j$, for all $p_j \in \text{part}(G)$.

**Proof.** By induction on the structure of $G$, using $\propto^\triangleright_\omega$ as in Def. A.7; see Appendix A.6. ▲

The above theorems offer a static guarantee that connects multiparty protocols and well-typed processes. They can be used to establish also dynamic guarantees relating the behavior of a global type $G$ and that of its associated set of multiparty systems (i.e., the typeful composition of $M^2[G](\Check{c})$ with processes for each of $p_j \in \text{part}(G)$). These dynamic guarantees can be easily obtained by combining Theorems 4.11 and 4.12 with the approach in [4].

5 Related Work

Our work contributes to the logical foundations of concurrency, a line of research developed by Caires and Pfenning [6], Dal Lago and Di Giamberardino [29], and others. Medium-based analyzes of multiparty sessions in a logical setting were developed in [4] and later used in an account of classical multiparty sessions in an extended linear logic [10].

Several process calculi with distributed features have been put forward. Two salient proposals are the Ambient calculus [12], in which processes may move across ambients—abstractions of administrative domains, and the distributed $\pi$-calculus (Dpi) [22], which
extends the π-calculus with flat locations, local communication, and process migration. While
domains in our model may be read as locations, this is just one specific interpretation for them;
their abstract and parametric nature admits various alternative readings (e.g. administrative
domains, security-related levels), leveraging the partial view of the domain hierarchy.

Concerning type systems for such calculi, e.g., those in [11, 3], typing is used to specify
security and communication-oriented properties in terms of ambient movement. Their work
does not cover issues of structured interaction, which is central in our work. Garralda et
al. [20] integrate binary sessions in an Ambient calculus, ensuring that session communication
is undisturbed by ambient mobility steps. This contrasts with our work, where typing ensures
that both migrations and communication are safe and, for the first time in such a setting,
satisfy global progress (i.e., session protocols never jeopardize migration steps and vice-versa).

Demangeon and Honda [14] study multiparty sessions with nested protocols. Their nesting
construct is similar to our global migration construct, which also introduces nesting. Their
work focuses on modularity in choreographic programming and is not concerned with domains
nor domain migration. As such, their nested protocols can have local participants and may
be parameterized on data from previous actions. We conjecture that our medium-based
approach can accommodate local participants in a similar way. Data parameterization can
be transposed to our logical setting via dependent session types [39, 42]. Asynchrony and
recursive behaviors can also be integrated by exploiting existing logical foundations [18, 41].

Balzer et al. [1] overlay a notion of world and accessibility on a system of shared session
types to ensure deadlock-freedom. Their work differs substantially from ours, being closer to
partial-order-based typing for deadlock-freedom [28, 33]: they instantiate accessibility as a
partial-order, equip sessions with multiple worlds and are not conservative wrt linear logic.

We highlight the works [21, 26, 15] that study runtime monitoring of contracts in session-based systems. Mediums can be seen as monitors that enforce the communication and
domain migrations specified in domain-aware multiparty sessions. The study of contract
enforcing mediums (e.g. enforcing refinements or limited trust) is interesting future work.

Finally, extensions of session types with access control and information flow analyses
have been proposed by Capecchi et al. [9, 8]. Our enforcement of communication across
connected domains introduces some high-level similarities with information flow analyses.
Establishing the precise relationship with such works is an interesting item of future work.

6 Concluding Remarks

We have proposed a Curry-Howard interpretation of hybrid linear logic as domain-aware
session types. Our development generalizes the interpretation put forward in [6], leading
to a typing discipline with enhanced expressiveness and strong correctness properties such
as global progress, termination, and session fidelity for well-typed domain-aware processes.
Domain-awareness is realized at both the process and type-level, accounting for scenarios
where domain information can be only determined at runtime. We also leverage a parametric
domain accessibility relation for added flexibility. Our system statically rules out processes
that communicate with unreachable domains, which is beyond the scope of previous works.

As an application of our framework, we presented the first systematic study of domain-awareness in a multiparty setting, considering multiparty sessions with domain-aware migration and communication whose semantics is given by a typed (binary) medium process that orchestrates the multiparty protocol. Embedded in a fully distributed domain structure, our
medium is shown to strongly encode domain-aware multiparty sessions; it naturally allows us
to transpose the correctness properties of our logical development to the multiparty setting.
References


13. Kaustuv Chaudhuri, Carlos Olarte, Elaine Pimentel, and Joëlle Despeyroux. Hybrid Linear Logic, revisited. *Mathematical Structures in Computer Science*, 2019. URL: [https://hal.inria.fr/hal-01968154](https://hal.inria.fr/hal-01968154).


A.1 Structural Congruence

Definition A.1. Structural congruence \((P \equiv Q)\) is the least congruence relation on processes such that

\[
P \equiv P' \quad P \equiv Q \Rightarrow P \equiv Q' \quad (\nu x)P \equiv (\nu y)Q \Rightarrow P' \equiv Q'
\]

A.2 Labeled Transition System

Some technical results rely on labeled transitions rather than on reduction. To characterize the interactions of a well-typed process with its environment, we extend the early labeled transition system (LTS) for the \(\pi\)-calculus [37] with labels and transition rules for choice, migration, and forwarding constructs. A transition \(P \xrightarrow{\lambda} Q\) denotes that \(P\) may evolve to \(Q\) by performing the action represented by label \(\lambda\). Transition labels are defined below:

\[
\lambda ::= \tau \mid x(y) \mid x(w) \mid x.l \mid x.y@\omega \mid x.y@\omega \mid x(y) \mid x(w) \mid x.l \mid x.y@\omega
\]

Actions are name input \(x(y)\), domain input \(x(w)\), the offers \(x.inl\) and \(x.inr\), migration \(x.y@\omega\) and their matching co-actions, respectively the output \(x(y)\) and bound output \(x@\omega\) actions, the domain output \(x@\omega\), label selections \(x.l\) and \(x.l\), and domain migration \(x.y@\omega\). Both the bound output \(x@\omega\) and migration action \(x.y@\omega\) denote extrusion of a fresh name \(y\) along \(x\).

Internal action is denoted by \(\tau\). In general, an action requires a matching co-action in the environment to enable progress.
### Domain-Aware Session Types

![Figure 2](image.png)

**Figure 2** Labeled Transition System.

#### Definition A.2 (Labeled Transition System).

The relation labeled transition \((P \xrightarrow{\lambda} Q)\) is defined by the rules in Fig. 2, subject to the side conditions: in rule (res), we require \(y \notin \text{fn}(\lambda)\); in rule (par), we require \(\text{bn}(\lambda) \cap \text{fn}(\lambda) = \emptyset\); in rule (close), we require \(y \notin \text{fn}(Q)\).

We omit the symmetric versions of rules (par), (com), and (close).

We write \(\text{subj}(\lambda)\) for the subject of the action \(\lambda\), that is, the channel along which the action takes place. Weak transitions are defined as usual. Let us write \(\rho_1 \rho_2\) for the composition of relations \(\rho_1, \rho_2\) and \(\Rightarrow\) for the reflexive, transitive closure of \(\xrightarrow{\lambda}\). Notation \(\xrightarrow{\lambda}\) stands for \(\xrightarrow{\lambda} \Rightarrow (\text{given } \lambda \neq \tau)\) and \(\xrightarrow{\tau}\) stands for \(\Rightarrow\). We recall basic facts about reduction, structural congruence, and labeled transition: closure of labeled transitions under structural congruence, and coincidence of \(\tau\)-labeled transition and reduction [37]: (1) if \(P \equiv \xrightarrow{\lambda} Q\) then \(P \xrightarrow{\lambda} \equiv Q\); (2) \(P \xrightarrow{\tau} Q\) if \(P \xrightarrow{\tau} \equiv Q\).
A.3 Omitted Typing Rules

\[
\begin{align*}
\text{(\&R)} & \quad \frac{\Omega ; \Gamma ; \Delta \vdash P_1 :: x:A_1[\omega] \quad \ldots \quad \Omega ; \Gamma ; \Delta \vdash P_n :: x:A_n[\omega]}{\Omega ; \Gamma ; \Delta \vdash \forall \{k : P_i\}_{i \in I} :: \exists \forall \{k : A_i\}_{i \in I}[\omega]} \\
\text{(\&L)} & \quad \frac{\Gamma ; \Delta, x:A[\omega] \vdash P :: x:C[\omega]}{\Gamma ; \Delta, x:A[\omega] \vdash P :: \exists C[\omega]} \\
\text{Lemma A.4} & \quad \frac{\Gamma ; \Delta, x:A[\omega] \vdash P :: \exists C[\omega]}{\Gamma ; \Delta, x:A[\omega] \vdash P :: x:C[\omega]} \quad k \not\in I \\
\text{Lemma A.5} & \quad \frac{\Gamma ; \Delta, x:A[\omega] \vdash P :: \exists C[\omega]}{\Gamma ; \Delta, x:A[\omega] \vdash P :: x:C[\omega]} \\
\text{(\&L)} & \quad \frac{\Omega ; \Gamma ; \Delta, x:A[\omega] \vdash Q_1 :: x:C[\omega] \quad \ldots \quad \Omega ; \Gamma ; \Delta, x:A[\omega] \vdash Q_n :: x:C[\omega]}{\Omega ; \Gamma ; \Delta, x:A[\omega] \vdash \forall \{Q_i\}_{i \in I} :: \exists C[\omega]} \\
\text{Lemma A.6} & \quad \frac{\Omega ; \Gamma ; \Delta \vdash Q :: x:A[\omega]}{\Omega ; \Gamma ; \Delta \vdash Q :: x:A[\omega]} \quad \Omega ; \Gamma ; \Delta \vdash \forall \{\nu x(y) : A[\omega]\} \vdash Q :: x:A[\omega] \\
\text{(cut')} & \quad \frac{\Omega ; \Gamma ; \Delta \vdash P :: x:A[\omega]}{\Omega ; \Gamma ; \Delta \vdash (\nu x)(\forall y)[(\nu x)(\forall y)P] :: x:A[\omega]}
\end{align*}
\]

A.4 Additional Lemmas for Type Preservation

The development of type preservation extends that of [6] to account for domain communication and migration. The proof mainly relies on a series of reduction lemmas (one per session type connective that produces observable process actions) that relate process actions with parallel composition through the (cut) rule, which correspond to logical proof reductions.

For instance, the reduction lemma for \( \ominus \) is:

\[ \text{Lemma A.3 (Reduction Lemma - } \ominus \text{). Assume} \]

\[ (a) \quad \Omega ; \Gamma ; \Delta \vdash P :: x:A_1 \ominus A_2[\omega] \quad \text{with } P \xrightarrow{\nu y \nu z} P' \quad \text{and} \]

\[ (b) \quad \Gamma ; \Delta_2, x:A_1 \ominus A_2[\omega] \vdash Q :: x:C[\omega'] \quad \text{with } Q \xrightarrow{\nu y \nu z} Q' \]

Then: \( \Omega ; \Gamma ; \Delta_2 \vdash (\nu x)(P' | Q') :: C[\omega'] \)

These lemmas carry over straightforwardly from [6]. The new lemmas are:

\[ \text{Lemma A.4 (Reduction Lemma - } \nu \text{). Assume} \]

\[ (a) \quad \Omega ; \Gamma ; \Delta \vdash P :: x:\nu \alpha .A[\omega] \quad \text{with } P \xrightarrow{x(u)} P' \quad \text{and} \]

\[ (b) \quad \Omega ; \Gamma ; \Delta_2, x:\nu \alpha .A[\omega] \vdash Q :: x:C[\omega] \quad \text{with } Q \xrightarrow{\nu y \nu z} Q' \]

Then: \( \Omega ; \Gamma ; \Delta_2 \vdash (\nu x)(P' | Q') :: C[\omega] \)

\[ \text{Lemma A.5 (Reduction Lemma - } \exists \text{). Assume} \]

\[ (a) \quad \Omega ; \Gamma ; \Delta \vdash P :: x:z \alpha .A[\omega] \quad \text{with } P \xrightarrow{t_{x(u)}} P' \quad \text{and} \]

\[ (b) \quad \Omega ; \Gamma ; \Delta_2, x:z \alpha .A[\omega] \vdash Q :: x:C[\omega] \quad \text{with } Q \xrightarrow{\nu y \nu z} Q' \]

Then: \( \Omega ; \Gamma ; \Delta_2 \vdash (\nu x)(P' | Q') :: C[\omega] \)

\[ \text{Lemma A.6 (Reduction Lemma - } \ominus \text{). Assume} \]

\[ (a) \quad \Omega ; \Gamma ; \Delta \vdash P :: x:@ u .A[\omega'] \quad \text{with } P \xrightarrow{x(u)} P' \quad \text{and} \]

\[ (b) \quad \Omega ; \Gamma ; \Delta_2, x:@ u .A[\omega'] \vdash Q :: x:C[\omega] \quad \text{with } Q \xrightarrow{x(u)} Q' \]

Then: \( \Omega ; \Gamma ; \Delta_2 \vdash (\nu x)(P' | Q') :: C[\omega'] \)
The proofs of the lemmas above follow by simultaneous induction on the two given typing derivations, with Lemmas A.4 and A.5 making use of Lemma 3.2. This development is essentially that of [5] and [39] which consider an extension of the core propositional system of [6] with communication of types (i.e., polymorphism) and communication of data (i.e., value dependencies). By appealing to such lemmas, we can establish type preservation for our system.

A.5 Pre-congruence on Local Types

The following definition is used in the proof of Thm. 4.12:

**Definition A.7.** We define $\preceq_\downarrow$ as the least pre-congruence relation on local types such that

$$T_1 \preceq_\downarrow T_1 \cup T_2 \quad T_1 \preceq_\downarrow T_2 \Downarrow \alpha.T \quad \text{if } \alpha \text{ does not occur in } T$$

A.6 Proofs of Medium Characterization

The proof of Theorem 4.11 relies on the following auxiliary proposition:

**Proposition A.8.** Let

1. $\Omega; \Gamma; \Delta, \nabla \vdash M^2[G_1](\overline{y}) : z : 1[\omega_m]$, with $\text{dom}(\Delta) = \{y_p, y_q_1, \ldots, y_q_n\}$
2. $\Omega; \Gamma; \Delta_1 \vdash y_p(m_p@ω_p).y_q_1(m_{q_1}@ω_{q_1}).\ldots.y_q_n(m_{q_n}@ω_{q_n}).M^2[G_2](\overline{m}) : z : 1[ω_m]$

be two compositional typings. Then

$$\Omega; \Gamma; \Delta_1 \circ \Delta_2 \vdash M^2[G_1](\overline{y}) \circ y_p(m_p@ω_p).y_q_1(m_{q_1}@ω_{q_1}).\ldots.y_q_n(m_{q_n}@ω_{q_n}).M^2[G_2](\overline{m}) : z : 1[ω_m]$$

is a compositional typing, where the typing environment $\Delta_1 \circ \Delta_2$ is defined as follows:

$$\Delta_1 \circ \Delta_2(c) = \begin{cases} \text{ } & \text{if } c \in \text{dom}(\Delta_1) \text{ and } c \not\in \text{dom}(\Delta_2), \text{ with } i, j \in \{1, 2\}, i \neq j \\ c : \langle \langle T_1 \circ T_2 \rangle \rangle[\omega] & \text{if } c : \langle \langle T_1 \rangle \rangle[\omega] \in \Delta_1 \text{ and } c : \langle \langle T_2 \rangle \rangle[\omega] \in \Delta_2 \end{cases}$$

**Proof (Sketch).** We must prove the existence of a typing derivation for the resulting fused process. We start by observing that the compositional typing for $M^2[G_1](\overline{y})$ ensures that its associated derivation will contain one or more occurrences of the sequent

$$\Omega; \Gamma; y_p : 1[ω_p], y_q_1 : 1[ω_{q_1}], \ldots, y_q_n : 1[ω_{q_n}] \vdash 0 : z : 1[ω_m] \quad (1)$$

corresponding to one or more occurrences of $\text{end}$ in $G_1$ (possible because of labeled choices in $G_1$). Indeed, by Def. 4.8 we have $M^2[\text{end}](\overline{y}) = 0$ and by Def. 4.10 we have $\langle \langle \text{end} \rangle \rangle = 1$. Observe that the compositional typing for

$$y_p(m_p@ω_p).y_q_1(m_{q_1}@ω_{q_1}).\ldots.y_q_n(m_{q_n}@ω_{q_n}).M^2[G_2](\overline{m}) \quad (2)$$

ensures that $\{y_p, y_{q_1}, \ldots, y_{q_n}\} \subseteq \text{dom}(\Delta_2)$, i.e., $\Delta_2$ contains judgements for at least the names in $\text{dom}(\Delta_1)$—it may also contain other judgments, corresponding to participants that intervene in $G_2$ but not in the sub-protocol $G_1$. Given this, the compositional typing for the fused process

$$M^2[G_1](\overline{y}) \circ y_p(m_p@ω_p).y_q_1(m_{q_1}@ω_{q_1}).\ldots.y_q_n(m_{q_n}@ω_{q_n}).M^2[G_2](\overline{m})$$

is obtained by “stacking up” the typing derivation for (2) exactly on the occurrences of sequents of the form (1) in the typing derivation for $M^2[G_1](\overline{y})$. This is fully consistent with definitions of fusion for processes (Def. 4.7) and local types (Def. 4.4): the former decrees that $0@P = P$ whereas the latter decrees that $\text{end} \circ T = T$. In the resulting “stacked” typing derivation, the types for $y_p, y_{q_1}, \ldots, y_{q_n}$ that correspond to the behavior of $M^2[G_1](\overline{y})$ can be derived exactly as in the derivation of the first assumption, now starting from the types $\Delta_2(y_p), \Delta_2(y_{q_1}), \ldots, \Delta_2(y_{q_n})$ rather than from 1. □
**Theorem 4.11** (Global Types → Typed Mediums). If $G$ is WF with $\part(G) = \{p_1, \ldots, p_n\}$ then $\Omega; \Gamma; c_p; \langle\langle G[p_1]\rangle\rangle[\omega], \ldots, c_p; \langle\langle G[p_n]\rangle\rangle[\omega] \vdash M^2[G](\bar{c}) : \top : 1[\omega_m]$ is a compositional typing, for some $\Omega, \Gamma$, with $\bar{\omega} = \omega_1, \ldots, \omega_n$. We assume that $\omega_i < \omega_m$ for all $i \in \{1, \ldots, n\}$ (the medium’s domain is accessible by all), and that $i \neq j$ implies $\omega_i \neq \omega_j$.

**Proof.** By induction on the structure of $G$. There are three cases.

- The base case, $G = \text{end}$, is immediate as there are no participants.
- The case $G = p_1 \rightarrow p_2; \{l_i(U_i).G^i\}_{i \in I}$ is exactly as in [4], but we report it here for the sake of completeness. By the well-formedness assumption (Def. 4.6), local types $G[p_1], \ldots, G[p_n]$ are all defined. Writing $p$ and $q$ instead of $p_1$ and $p_2$, by Def. 4.5 we have:
  \[
  G[p] = p\{l_i(U_i).G^i[p]\}_{i \in I} \tag{3}
  \]
  \[
  G[q] = p\{l_i(U_i).G^i[q]\}_{i \in I} \tag{4}
  \]
  \[
  G[p_j] = \bigcup_{i \in I} G^i[p_j] \quad \text{for every } j \in \{3, \ldots, n\} \tag{5}
  \]
  We need to show that, for some $\Omega$ and $\Gamma$,
  \[
  \Omega; \Gamma; c_p; \langle\langle G[p]\rangle\rangle[\omega_p], c_q; \langle\langle G[q]\rangle\rangle[\omega_q], \Delta \vdash M^2[G](\bar{c}) : \top : 1[\omega_m] \tag{6}
  \]
  is a compositional typing, with $D = c_p; \langle\langle G[p_1]\rangle\rangle[\omega_{p_1}], \ldots, c_p; \langle\langle G[p_n]\rangle\rangle[\omega_{p_n}]$.

  Without loss of generality, we detail the case $I = \{1, 2\}$. By Def. 4.8, we have:
  \[
  M^2[G](\bar{c}) = c_p \left\{ l_1 : c_p(u).c_q < l_1; c_\bar{v}(v).([u \leftrightarrow v] \mid M^2[G^1](\bar{c})) \right. \\
  l_2 : c_p(u).c_q < l_2; c_\bar{v}(v).([u \leftrightarrow v] \mid M^2[G^2](\bar{c})) \right. 
  \]
  and by combining (3) and (4) with Def. 4.10 we have:
  \[
  \langle\langle G[p]\rangle\rangle = \bigoplus \{ l_1 : \langle\langle U_1\rangle\rangle \otimes \langle\langle G^1[p]\rangle\rangle, l_2 : \langle\langle U_2\rangle\rangle \otimes \langle\langle G^2[p]\rangle\rangle \}_{i \in I} \tag{7}
  \]
  \[
  \langle\langle G[q]\rangle\rangle = \bigoplus \{ l_1 : \langle\langle U_1\rangle\rangle \otimes \langle\langle G^1[q]\rangle\rangle, l_2 : \langle\langle U_2\rangle\rangle \otimes \langle\langle G^2[q]\rangle\rangle \}_{i \in I} \tag{8}
  \]
  Now, by assumption $G$ is WF; then, by construction, both $G^1$ and $G^2$ are WF too.

  Therefore, by using IH twice we may infer that both
  \[
  \Omega; \Gamma; c_p; \langle\langle G^1[p]\rangle\rangle[\omega_p], c_q; \langle\langle G^1[q]\rangle\rangle[\omega_q], \Delta_1 \vdash M^2[G^1](\bar{c}) : \top : 1[\omega_m] \tag{7}
  \]
  \[
  \Omega; \Gamma; c_p; \langle\langle G^2[p]\rangle\rangle[\omega_p], c_q; \langle\langle G^2[q]\rangle\rangle[\omega_q], \Delta_2 \vdash M^2[G^2](\bar{c}) : \top : 1[\omega_m] \tag{8}
  \]
  are compositional typings, for any $\Omega$ and $\Gamma$, with
  \[
  \Delta_1 = c_p; \langle\langle G^1[p_1]\rangle\rangle[\omega_{p_1}], \ldots, c_p; \langle\langle G^1[p_n]\rangle\rangle[\omega_{p_n}] \tag{7}
  \]
  \[
  \Delta_2 = c_p; \langle\langle G^2[p_1]\rangle\rangle[\omega_{p_1}], \ldots, c_p; \langle\langle G^2[p_n]\rangle\rangle[\omega_{p_n}] \tag{8}
  \]
  Now, to obtain a compositional typing for $M^2[G](\bar{c})$, we must consider that $\Delta_1$ and $\Delta_2$ may not be identical. This is due to the merge-based well-formedness assumption, which admits non identical behaviors in branches $G^1$ and $G^2$ in the case of (local) branching types (cf. Def. 4.3).

  We proceed by induction on $k$, defined as the size of $\Delta_1$ and $\Delta_2$ (note that $k = n - 2$).

  1. (Case $k = 0$): Then $\Delta_1 = \Delta_2 = \emptyset$ and $p$ and $q$ are the only participants in $G$. Let us write $A_q$ to stand for the session type
  \[
  \&\{ l_1 : \langle\langle U_1\rangle\rangle \otimes \langle\langle G^1[q]\rangle\rangle, l_2 : \langle\langle U_2\rangle\rangle \otimes \langle\langle G^2[q]\rangle\rangle \}
  \]
Based on (7) and (8), following the derivation in Fig. 3, we may derive typings

Ω; Γ; c_p ∶ (U_1) ⊕ (G^1)p_1; c_q ∶ (U_1) ⊕ (G^1)p_2; c_r ∶ (U_1) ⊕ (G^1)p_3; c_s ∶ (U_1) ⊕ (G^1)p_4; (u ↦ v) ∶ M^2 [G^1](c) :: z ∶ 1[w_m]

(9)

Ω; Γ; c_p ∶ (U_1) ⊕ (G^2)p_1; c_q ∶ (U_1) ⊕ (G^2)p_2; (u ↦ v) ∶ M^2 [G^2](c) :: z ∶ 1[w_m]

(10)

The proof for this case is completed using (9) and (10) as premises for Rule (⊕L):

Ω; Γ; c_p ∶ (l_1 ∶ (U_1) ⊕ (G^1)p_1; l_2 ∶ (U_1) ⊕ (G^2)p_2) [w_p], c_q ∶ (A_q; A_q) ⊕ M^2 [G^1](c) :: z ∶ 1[w_m]

(11)

2. (Case k > 0): Then there exists a participant p_k, types B_1 = (G^1)p_k, B_2 = (G^2)p_k and environments Δ_1', Δ_2' such that Δ_1 = c_p; B_1[ω_p], Δ_1' and Δ_2 = c_p; B_2[ω_p], Δ_2'.

By induction hypothesis, there is a compositional typing starting from

Ω; Γ; c_p; (G^1)p_1 [w_p], c_q; (G^2)p_2 [w_q], (m ↦ n) ∶ M^2 [G^1](c) :: z ∶ 1[w_m]

(12)

resulting into

Ω; Γ; c_p; (l_1 ∶ (U_1) ⊕ (G^1)p_1; l_2 ∶ (U_1) ⊕ (G^2)p_2) [w_p], c_q; (A_q; A_q), (m ↦ n) ∶ M^2 [G^1](c) :: z ∶ 1[w_m]

since Δ_1' = Δ_2'.

To extend the typing derivation to Δ_1 and Δ_2, we proceed by a case analysis on the shape of B_1 and B_2. We aim to show that either (a) B_1 and B_2 are already identical base or session types or (b) that typing allows us to transform them into identical types. We rely on the definition of ⊕ (Def. 4.3). There are two main sub-cases:

a. Case B_1 ≠ (l_h ∶ A_h) ∈ H: Then, since Def. 4.3 decrees T ∪ T = T and the fact that merge-based well-definedness depends on L, we may infer B_2 = B_1. Hence, Δ_1 = Δ_2 and the desired derivation is obtained as in the base case.

b. Case B_1 = (l_h ∶ A_h) ∈ H: This is the interesting case: even if merge-based well-formedness of G ensures that both B_1 and B_2 are selection types, they may not be identical. If B_1 and B_2 are identical then we proceed as in previous sub cases. Otherwise, then due to L there are some finite number of labeled alternatives in B_1 but not in B_2 and/or viceversa. Also, Def. 4.3 ensures that common options (if any) are identical in both branches. We may then use Rule (∀L) (cf. Appendix A.3) to “complement” occurrences of types B_1 and B_2 in (7) and (8) as appropriate to make them coincide and achieve identical typing. Rule (∀L) is silent; as labels are finite, this completing task is also finite, and results into Δ_1 = Δ_2.
Finally, we have the case $G = p$ moves $q_1, \ldots, q_n$ to $w$ for $G_1; G_2$. Without loss of generality, we consider the global type $G = p$ moves $q$ to $w$ for $G_1; G_2$, i.e., the type in which the sub-protocol $G_1$ only involves two participants, namely $p$ and $q$. By the well-formedness assumption, local types $G_1[p]$, $G_1[q]$, $G_1[p_3], \ldots, G_1[p_m]$ are all defined. In particular, by Def. 4.5 we have:

$$G_1[p] = \downarrow.\beta.(\exists x.\alpha_0.G_1[p] \circ \alpha_0.G_2[p])$$

(11)

$$G_1[q] = \downarrow.\beta.(\forall x.\alpha_0.G_1[q] \circ \alpha_0.G_2[q]$$

(12)

We need to show that, for some $\Gamma$,

$$\Omega; \Gamma; c_p; \langle G_1[p] \rangle[\omega_0], c_q; \langle G_1[q] \rangle[\omega_0], \Delta[\emptyset/m] \vdash M^2[G]\langle c \rangle :: z : 1[\omega_m]$$

(13)

with $\Delta[\emptyset/m] = c_{p_3}; \langle G_1[p_3] \rangle[\omega_3], \ldots, c_{p_n}; \langle G_1[p_n] \rangle[\omega_n]$ is a compositional typing. By Def. 4.8, we have:

$$M^2[G]\langle c \rangle = c_p(\alpha).c_q(\alpha).c_q(y_q \otimes \alpha).c_q(y_q \otimes \omega).$$

$$M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(14)

and by combining (11) and (12) with Def. 4.10 we have:

$$\langle G_1[p] \rangle = \downarrow.\beta.\exists x.\alpha_0.G_1[p] \circ \alpha_0.G_2[p]$$

(15)

$$\langle G_1[q] \rangle = \downarrow.\beta.\forall x.\alpha_0.G_1[q] \circ \alpha_0.G_2[q]$$

(16)

Now, by assumption $G$ is WF; then, by construction both $G_1$ and $G_2$ are WF too.

Therefore, by using IH twice we may infer that both

$$\Omega; \Gamma; y_p; \langle G_1[p] \rangle[\alpha], y_q; \langle G_1[q] \rangle[\alpha] \vdash M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(17)

Using Prop. A.8 on (15) and (17) we obtain:

$$\Omega; \Gamma; y_p; \langle G_1[p] \circ \alpha_0. G_2[p] \rangle[\alpha], y_q; \langle G_1[q] \circ \alpha_0. G_2[q] \rangle[\alpha], \Delta \vdash M^2[G_2][\emptyset/m]$$

(18)

We then have the following derivation, which completes the proof for this case:

$$\Omega; \omega_1 \prec \alpha, \omega_2 \prec \alpha; \Gamma; y_p; \langle G_1[p] \circ \alpha_0. G_2[p] \rangle[\alpha], c_q(\alpha_0).\langle G_1[q \circ \alpha_0. G_2[q] \rangle[\omega_2], \Delta$$

(6L)

$$\vdash c_q(y_q \otimes \alpha_0).M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(6L)

$$\Omega; \omega_1 \prec \alpha, \omega_2 \prec \alpha; \Gamma; c_q(\alpha_0).\langle G_1[p \circ \alpha_0. G_2[p] \rangle[\omega_1], c_q(\alpha_0).\langle G_1[q \circ \alpha_0. G_2[q] \rangle[\omega_2], \Delta$$

(8L)

$$\vdash c_q(y_q \otimes \alpha_0).c_q(y_q \otimes \alpha_0).M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(8L)

$$\Omega; \omega_1 \prec \alpha; \Gamma; c_q(\alpha_0).\langle G_1[p \circ \alpha_0. G_2[p] \rangle[\omega_1], c_q(\alpha_0).\langle G_1[q \circ \alpha_0. G_2[q] \rangle[\omega_2], \Delta$$

(3L)

$$\vdash c_q(\alpha_0).c_q(\alpha_0).c_q(y_q \otimes \alpha_0).c_q(y_q \otimes \alpha_0).M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(3L)

$$\Omega; \Gamma; c_q(\alpha_0).\langle G_1[p \circ \alpha_0. G_2[p] \rangle[\omega_1], c_q(\alpha_0).\langle G_1[q \circ \alpha_0. G_2[q] \rangle[\omega_2], \Delta$$

(\|L)

$$\vdash c_q(\alpha_0).c_q(\alpha_0).c_q(\alpha_0).c_q(y_q \otimes \alpha_0).c_q(y_q \otimes \alpha_0).M^2[(\alpha/\omega/\alpha/\omega)[G_1][y] \circ y_q(m_q \otimes \omega_p).y_q(m_q \otimes \omega_q)].M^2[G_2][\emptyset/m]$$

(\|L)_2

The proof of the converse of Thm. 4.11 proceeds similarly; we must take into account that, given a global type $G$, the process structure of $M^2[G]\langle c \rangle$ will induce types closely related to $G_1[p_1], \ldots, G_1[p_m]$, up to occurrences of two type operators whose typing rules enforce a silent interpretation of processes, namely $(kL_2)$ and $(L)$. 
After the negotiation stage is complete, both the client and the instrument migrate to a domain in which the agent and the client coexist in order to successfully negotiate the instrument usage.

This example is adapted from [14], consisting of a negotiation procedure \textit{Nego} between two participants of a three-party interaction. The negotiation consists of an agreement on a contract: one participant specifies a request, while the other offers a corresponding contract. The first participant may either accept the contract and end the protocol or make a counter-offer. For the sake of conciseness we assume that the counter-offer is accepted:

\[
\begin{array}{c}
\text{Neg}_{\rho, q} \triangleq p \rightarrow q: \{\text{ask(terms)}, q \rightarrow p: \{\text{proposition(contract)}_1, p \rightarrow q: \{\text{accept.end, counter(contract)}_1, q \rightarrow p: \{\text{accept.end}\}\}\}\}\end{array}
\]

The main protocol consists of a client, an agent and an instrument, each initially in their own domains. The client first sends a request to the agent for some instrument they wish to use. The agent connects to the instrument which acknowledges when available. The agent then enters the negotiation sub-protocol with the client (via protocol \textit{Nego}), by having both agent and the client migrate to domain \(d_u\). This movement models the trusted setting at which the agent and the client coexist in order to successfully negotiate the instrument usage. After the negotiation stage is complete, both the client and the instrument migrate to a common domain \(d_t\) to perform the rest of the protocol, which for the sake of conciseness we model with the client either aborting the interaction or sending a command to the instrument and then receiving back the appropriate result:

\[
\begin{array}{c}
\text{Neg}_{\rho, q} \triangleq p \rightarrow q: \{\text{ask(terms)}, q \rightarrow p: \{\text{proposition(contract)}_1, p \rightarrow q: \{\text{accept.end, counter(contract)}_1, q \rightarrow p: \{\text{accept.end}\}\}\}\}\end{array}
\]
By leveraging our notion of medium, we can make explicit the fact that the three participants are distributed agents, each located at independent domains that can access the medium substrate (i.e. the domain of the medium). Through the medium-orchestrated interaction, the use of domain migration primitives enables us to explicitly model the various domain movement steps that the participants must follow to implement the protocol. This is in sharp contrast with more traditional approaches to multiparty protocols [25], where such domain specific notions are implicit. The medium for the global type above is (we assume the three participants initially reside at worlds \( w_{\text{client}}, w_{\text{agent}}, w_{\text{instr}} \), respectively):

\[
\begin{align*}
\text{client} \rightarrow \text{agent} : \{ & \text{req(coord), agent} \rightarrow \text{instr}: \{ \text{connect, instr} \rightarrow \text{agent}: \{ \text{available, agent} \rightarrow \text{client}: \{ \text{ack.} \} \} \} \} ; \\
\text{client} \rightarrow \text{instr} : \{ & \text{move client to } d_u \text{ for Nego}_{\text{agent,client}}, \text{client} \rightarrow \text{Instr} : \{ \text{abort.end, command(code), instr} \rightarrow \text{client}: \{ \text{result(data), end } \} \} \} ; \end{align*}
\]

The first two lines of the medium definition correspond to the initial exchange between the client, the agent and the instrument. The actions after the emission of label \text{ack} to the client model the migration protocol: the agent emits the domain identifier to the medium, which then forwards it to the client and receives from both participants the session handles \( y_{\text{agent}} \) and \( y_{\text{client}} \), located at \( d_u \). After the migration takes place, the medium orchestrates the negotiation between the agent and the client using the new session handles.

After the negotiation, the agent and the client migrate back to their initial domains \( w_{\text{agent}} \) and \( w_{\text{client}} \), respectively, and the interactions between the client and the instrument take place: the client and the instrument migrate to \( d_i \), sending to the medium the session handles \( y_{\text{client}} \) and \( y_{\text{instr}} \), followed by the client emitting an \text{abort} or \text{command} message which is forwarded to the instrument. In the latter case, the instrument forwards the result to the client. Finally, both the client and the instrument migrate back to their initial domains.

### B.2 Domain-aware Middleware

A common design pattern in distributed computing is the notion of a middleware agent which answers requests from clients, sometimes offloading the requests to some server (e.g. to better manage local resource availability). The mediation between the middleware and the server often involves some form of privilege escalation or specialized authentication, which we can now model via domain migration. We first represent a simple offloading protocol between the middleware \( p \) and the server \( q \):

\[
\text{Offload}_{p,q} \triangleq p \rightarrow q: \{ \text{req(data), q} \rightarrow p: \{ \text{reply(ans), end} \} \}
\]

The global interaction is represented by the following global type:

\[
\begin{align*}
\text{client} \rightarrow \text{mw} : \{ \text{request(req)} \} ; \\
\text{mw} \rightarrow \text{client} : \{ \text{reply(ans)} \} ; \\
\text{mw} \rightarrow \text{server} : \{ \text{done.end} \} ; \\
\text{wait} \text{mw} \rightarrow \text{server} : \{ \text{init.mw moves server to } w_{\text{priv}} \text{ for } \text{Offload}_{\text{mw,server}} ; \\
\text{mw} \rightarrow \text{client} : \{ \text{reply(ans), 0} \} \}
\end{align*}
\]
The medium for this protocol is given by:

\[
c_{\text{client}} \triangleright \{\text{request} : c_{\text{client}}(r) \triangleright \text{addCart} \triangleright \text{cc} \triangleright \{\text{done} : c_{\text{server}} \triangleleft \text{doDone} \triangleleft 0\}\},
\]

\[
c_{\text{store}} \triangleright \{\text{reply} : c_{\text{store}}(a) \triangleright \text{cc} \triangleright \text{bank} \triangleright \{\text{done} : c_{\text{server}} \triangleleft \text{doDone} \triangleleft 0\}\},
\]

\[
\text{wait} : c_{\text{store}} \triangleright \{\text{init} : c_{\text{server}} \triangleleft \text{init} ; c_{\text{store}}(w_{\text{priv}}) ; c_{\text{server}}(w_{\text{priv}}) ; c_{\text{store}}(\text{cc}) ; c_{\text{store}}(\text{bank}) \},
\]

\[
c_{\text{store}}(\text{addCart} ; c_{\text{store}}(\text{cc}) ; c_{\text{store}}(\text{bank}) ; c_{\text{server}}(w_{\text{priv}}) ; c_{\text{server}}(w_{\text{priv}}) ; M_{\text{priv}} \langle \text{Offload} \rangle (y_{\text{store}} ; y_{\text{server}}) \}
\]

\[
\text{z}_{\text{store}} \triangleright \{\text{reply} : z_{\text{store}}(a) \triangleright \text{cc} \triangleright \text{bank} \triangleright \{\text{done} : (a \leftrightarrow n) \mid 0\}\}\\}
\]

Notice how the client’s domain remains fixed throughout the entire interaction, regardless of whether or not the middleware chooses to interact with the server to fulfil the client request.

### B.3 A Secure Communication Domain

Our previous examples explore the use of domains in a general distributed setting. Another interesting aspect of our domain-aware typing discipline is that communication can only take place between reachable domains. In scenarios where participants are each situated in distinct domains, domain movement also governs the ability of participants to interact. For instance, consider the following protocol excerpt:

\[
\text{client} \rightarrow \text{store} : \{\text{purchase} : \text{store moves bank to sec for SecurePay} ;
\]

\[
\text{store} \rightarrow \text{client} : \{\text{success (receipt). end. fail. end}\}\}
\]

The protocol above is part of the interaction between an online store and its clients, where after some number of exchanges the client decides to purchase the contents of their shopping cart. Upon receiving the purchase message, the store is meant to enter a secure domain so it can communicate with the bank role to exchange potentially sensitive data. In a setting where the client, store and bank exist at different domains and where only the store domain can reach the bank domain, our domain-aware typing discipline ensures that no direct communication between the bank and the client domain is possible, with the entirety of the data flows between the client and the bank (via the medium) being captured by the type and medium specification.

### C Examples of Domain-aware Binary Sessions

In this section we present two examples that further illustrate the novel features of our domain-aware framework for session communications.

#### C.1 E-Commerce Example

We revisit the web store example of § 3. Recall the refined web store session type:

\[
\text{WStore}_{\text{sec}} \triangleq \text{addCart} \rightarrow & \{\text{buy} : @\text{sec} \text{Payment}. \text{quit} : 1\}
\]

\[
\text{Payment} \triangleq \text{CCNumber} \rightarrow \oplus \{\text{ok} : (@\text{bankReceipt}) \oplus 1, \text{nok} : 1\}
\]

\[
\text{Bank} \triangleq \text{CCNumber} \rightarrow \oplus \{\text{ok} : \text{Receipt} \oplus 1, \text{nok} : 1\}
\]

The process that implements the bank interface is to be accessible from the domain of the web store, moving to a secure domain \text{bank} before receiving and validating the payment information. Thus, the bank process typing can be specified with the following judgment, recalling that the domain of the web store is \text{ws}:

\[
\text{ws} \leftarrow \text{bank} ; (; \vdash B : @\text{bank} \text{Bank}[\text{ws}]
\]
The web store will then use the bank interface to fulfill its interface:

\[
\vdash b : @\text{bank}\text{Bank}\{\text{ws}\} \Rightarrow \text{Store} :: z : W\text{Store}_{\text{sec}}\{\text{ws}\}
\]

The \(\triangleleft\) type constructor allows us to express a very precise form of coordinated domain migration. For instance, typing ensures that in order for the store to produce an output of the form \(\triangleright \text{bank}\text{Payment}\) it must first have interacted with the bank domain: in order to produce an output of \(\triangleright \text{bank}\), it must be the case that \(\text{ws} \prec \text{bank}\), which is only known to the store process after interacting with the bank domain. Alternatively, consider the \(W\text{Store}_{\text{sec}}\text{service}\) interacting with a client process along channel \(x\), each in their own (reachable) domains, \(c\) and \(\text{ws}\), respectively. Our framework ensures that interactions between the client and the web store enjoy session fidelity, progress, and termination guarantees. Concerning domain-awareness, by assuming the client chooses to buy his product selection, we reach a state that is typed as follows:

\[
c \prec \text{ws}; :: x : \triangleright \text{sec}\text{Payment}\{\text{ws}\} \Rightarrow \text{Client} :: z : \triangleright \text{sec}\{c\}
\]

At this point, it is \textit{impossible} for a (typed) client to interact with the behavior that is protected by the trusted domain \(\text{sec}\), since it is not the case that \(c \prec^* \text{sec}\). This ensures, e.g., that a client cannot exploit the payment platform of the web store by accessing the trusted domain in unforeseen ways. Formally, no typing derivation of \(c \prec \text{ws}; :: \text{Payment}\{\text{sec}\} \Rightarrow \text{Client} :: z : \triangleright \text{sec}\{c\}\) exists (Theorem 3.6). The client can only communicate in the secure domain after the web store service has migrated accordingly:

\[
c \prec \text{ws}, \text{ws} \prec \text{sec}; :: x' : \triangleright \text{sec}\text{Payment}\{\text{sec}\} \Rightarrow \text{Client'} :: z' : \triangleright \text{sec}\{c\}
\]

where \(\text{Client'} \triangleq x(x'@\text{sec}).\exists(z'@\text{sec}).\text{Client'}\).

It is inconvenient (and potentially error-prone) for the payment domain to be hardwired in the type. We can solve this issue via existential quantification as shown in the introduction.

\[
W\text{Store}_3 \triangleq \text{addCart} \Rightarrow \{\text{buy} : \exists \alpha. \triangleright \alpha\text{Payment}, \text{quit} : 1\}
\]

As long as accessibility is irreflexive and antisymmetric, the server-provided payment domain \(w\) will not be able to interact with the initial public domain of the interaction except as specified in the \(\text{Payment}\) type.

Alternatively, the server can let the client choose a payment domain by using universal quantification. Compliant server code will only be able to communicate in the client-provided payment domain since the process must be parametric in \(\alpha\).

\[
W\text{Store}_v \triangleq \text{addCart} \Rightarrow \{\text{buy} : \forall \alpha. \triangleright \alpha\text{Payment}, \text{quit} : 1\}
\]

### C.2 Spatial Distribution as in λ5

Murphy et al. [32] have proposed a Curry-Howard interpretation of the intuitionistic modal logic S5 [38] to model distributed computation with worlds as explicit loci for computation. Accessibility between worlds was assumed to be reflexive, transitive, and symmetric because each host on a network should be reachable from any other host. Murphy [31] later generalized this to hybrid logic, an idea also present in [27], so that propositions can explicitly refer to worlds. Computation in this model was decidedly sequential, and a concurrent extension was proposed as future work. Moreover, the system presented some difficulties in the presence of disjunction, requiring a so-called \textit{action at a distance} without an explicit communication visible in the elimination rule for disjunction.
The present system not only generalizes λ5 to permit concurrency through session-typed linearity, but also solves the problem of action at a distance because all communication is explicit in the processes. Due to this issue in the original formulation of λ5, we will not attempt here to give a full, computationally adequate interpretation of λ5 in our system (which would generalize [40]), but instead explain the spatially distributed computational interpretation of our type system directly.

- c : □A. Channel c offering A can be used in any domain.
- c : ◊A. Channel c is offering A in some (hidden) domain.

These are mapped into our hybridized linear logic (choosing a fresh α each time) with

\[ \square A = \forall \alpha, @_\alpha A \]
\[ \Diamond A = \exists \alpha, @_\alpha A \]

A process \( P ::= c.\square A[w_1] \) will therefore receive, along c, a world \( w_2 \) reachable from \( w_1 \) and then move to \( w_2 \), offering A in domain \( w_2 \). Conversely, a process \( P ::= c.\Diamond A[w_1] \) will send a world \( w_2 \) along c and then move to \( w_2 \), offering A in domain \( w_2 \). Processes using such channels will behave dually.

We can now understand the computational interpretation of some of the distinctive axioms of S5, keeping in mind that accessibility should be reflexive, transitive, and symmetric (we make no distinction between direct reachability or reachability requiring multiple hops).

- \( K_\Diamond ::= z.\square(A \rightarrow B) \rightarrow \Diamond A \rightarrow \Diamond B[w_0] \).

Here, A is offered along some \( c_1 \) at some unknown domain \( \alpha \) reachable from \( w_0 \). Move the offer of \( \square(A \rightarrow B)[w_0] \) to \( \alpha \) as \( c_2 \) and send it \( c_1 \) to obtain \( c_2 : B[\alpha] \). We abstract this as \( \Diamond B[w_0] \), which is possible since \( w_0 \preceq \alpha \). Intuitively, this axiom captures the fact that a session transformer \( A \rightarrow B \) that can be used in any domain may be combined with a session A offered in some domain to produce a session behavior B, itself in some hidden domain.

\[ K_\Diamond \triangleq z(x).z(y).g(\alpha).g(c_1@\alpha).x(\alpha).x(c_2@\alpha).z(c_2 \leftrightarrow c_3) \]

- \( T ::= z.\square A \rightarrow A[w_0] \).

Given a session \( x \) that offers \( \square A \) at \( w_0 \), we offer A at \( w_0 \) by appealing to reflexivity, and thus \( w_0 \preceq w_0 \). This means that we can then receive from \( x \) a fresh channel \( c_1@w_0 \), obtaining an ambient session of type A at domain \( w_0 \) which we then forward along z. The T axiom captures the fact that a mobile session can in fact be accessed “anywhere”.

\[ T \triangleq z(x).x(w_0).x(c_1@w_0).[c_1 \leftrightarrow z] \]

- \( 5 ::= z.\Diamond A \rightarrow \Diamond A[w_0] \)

Given a session \( x \) offering \( \Diamond A \) at \( w_0 \), this means that \( x \) is offering the behavior A at some reachable but unknown domain \( \alpha \). We can use this session to provide somewhat of a “link” session, that allows any other domain \( \alpha \) to also reach the behavior A at this reachable, unknown domain. We do this by first receiving \( \alpha \) along z, identifying the domain reachable from \( w_0 \) which we wish to link to \( \beta \). We then send along z the fresh session \( c_1 \) located at \( \alpha \), along which we shall provide the connection. We may then receive from x the identity of \( \beta \) and a session \( c_2 \) located in this domain, send the identity along \( c_1 \) and a fresh session \( c_3 \), located at \( \beta \) (only possible due to the accessibility relation being an equivalence) which we then forward from \( c_2 \) as needed.

\[ 5 \triangleq z(x).z(\alpha).z(c_1@\alpha).x(\beta).x(c_2@\beta).c_1(\beta).c_1(c_3@\beta).[c_2 \leftrightarrow c_3] \]

Other axioms can be given similarly straightforward interpretations and process realizations.