Logic-Based Domain-Aware Session Types

Luís Caires*, Jorge A. Pérez*, Frank Pfenning† and Bernardo Toninho*†
*FCT - Universidade Nova de Lisboa
†Carnegie Mellon University

***Abstract*—Software services and governing communication protocols are increasingly domain-aware. Domains can have multiple interpretations, such as the principals on whose behalf processes act or the location at which parties reside. Domains impact protocol compliance and access control, two central issues to overall functionality and correctness in distributed systems.**

This paper proposes a session-typed process framework for domain-aware communication-centric systems based on a Curry-Howard interpretation of linear logic, here augmented with nominals from hybrid logic indicating domains. These nominals are explicit in the process expressions and govern domain migration, subject to a parametric accessibility relation familiar from the Kripke semantics for modal logic. Flexible access relationships among domains can be elegantly defined and statically enforced. The framework can also account for scenarios in which domain information is discovered only at runtime.

Due to the logical origins of our systems, well-typed processes enjoy session fidelity, global progress, and termination. Moreover, well-typed processes always respect the accessibility relation and satisfy a form of domain parametricity, two properties crucial to show that domain-related properties of concrete programs are satisfied.

I. Introduction

Modern software systems rely on *communication* between heterogeneous services. Overall system correctness depends on compliance to service protocols, which often combine advanced forms of mobility, concurrency, and distribution. As services are nowadays offered virtualized in third-party platforms (e.g., cloud computing), structured communications should span diverse software and hardware *domains*. As a consequence, communication is increasingly *domain-aware*, in the sense that a partner's potential for interaction is influenced by the domains in which it is involved in at various points of a protocol. As partners frequently reside in different domains, *connectedness* among domains plays a vital role in enabling communications. Moreover, domain architectures are rarely fully specified: to aid modularity and platform independence system participants such as service developers, platform vendors, or service clients often have only local and/or partial views of actual domain structures. Despite the importance of domains for protocol compliance, access control, and security, its formal, logical status within models of communication-centric systems remains largely unexplored.

This paper contributes to typed approaches to the analysis of domain-aware structured communications, with a focus on *session-based concurrency*. In this approach, concurrent dialogues are structured into basic units called *sessions*; interaction patterns are abstracted as *session types* [1], [2], [3], against which process specifications may be checked. As these specifications are usually given in the π-calculus [4], we obtain processes interacting on so-called session channels which connect exactly two subsystems. Communication correctness is usually characterised by the combination of two properties: *session fidelity* (type preservation) and *freedom from deadlock* (progress). The former states 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). Some disciplines also ensure that well-typed processes are *terminating* (strongly normalizing). Intuitively, this guarantees responsive partners which never exhibit infinite internal behavior.

As a simple example, consider the session type WStore below. It abstracts a typical interaction sequence between an online store and its clients, seen from the store's perspective:

$$\mathsf{WStore} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \mathsf{Payment}\,,\ \mathtt{quit} : \mathbf{1}\}$$
$$\mathsf{Payment} \triangleq \mathsf{ShipAddress} \multimap \mathsf{CCNumber} \multimap \mathsf{Receipt} \otimes \mathbf{1}$$

WStore specifies a service that awaits an input (denoted by $\multimap$) of items the client wishes to add to his shopping cart. After some exchanges (elided), the store presents the client with a choice (denoted with a labelled $\&$) to either buy the selected items or quit the session. If the client chooses to buy, the store requires shipping and credit card information, after which it emits (denoted by $\otimes$) a receipt and closes the interaction ($\mathbf{1}$).

A main motivation for this paper is the contrast between (a) the growing relevance of domain-awareness in real communicating systems and (b) the expressiveness of existing session types frameworks, which appear inadequate to specify and enforce requirements related to domain-awareness. These limitations may hinder the effectiveness of the analysis techniques derived from these theories. Even a small interface such as WStore witnesses these limitations: although it captures the *communication behavior* intended for this service, its high-level of abstraction is insufficient in realistic scenarios. In fact, online stores are expected to request the client's sensitive information only after entering a *trusted domain* (e.g., by establishing an HTTPS connection). Dually, the online store should not allow insecure client accesses to its payment infrastructure. Unfortunately, these domain-specific details are very hard to express appropriately in existing session type systems. As type structures are typically defined with particular process models in mind, expressiveness issues at the level of types are often reflected also at the level of processes. As a result, usual typed process languages do not capture domain-awareness in specifications. All of this calls for a typed framework in which structured communication and domain-awareness issues are treated in a unified way, relying on carefully designed integrations of richer type structures with suitably extended process languages.

This paper contains four main contributions. The first is a process model with explicit domain-based migration (§ II). We consider a session π-calculus enriched with domain information that can be exchanged by processes and domain movement prefixes $x\langle y@w\rangle.P$ and $x(y@w).Q$. Intuitively, $x\langle y@w\rangle.P$ denotes a process that is prepared to migrate the communication actions in P offered along x to a channel y that is located at domain w. Typing will ensure that y is fresh, so this is used as a bound output. A process $x(y@w).Q$ is complementary: it signals a session x to migrate to domain w by continuing communication along a (fresh) channel y it receives. These two prefixes may synchronize to realize coordinated migration to w with continuations P and Q.

Our second main contribution is a logic-based session-typed discipline over domain-aware communicating processes (§ IV). By relying on a variant of intuitionistic linear logic with features from *hybrid logic* [5] (HILL, defined in § III), we generalize an interpretation of linear logic propositions as session types given in prior work [6]. There are two key aspects. First, we interpret a *(modal) world* as a *domain* where session behavior resides. As a result, judgments in our system not only describe the services that a process requires to implement a given session behavior (as in [6]), but they also stipulate the domains in which such services should be present. Second, the hybrid connective $@_w$ of HILL is interpreted as a *migration step* to domain/world w, which at the level of π-calculus processes is made explicit by referencing w itself in combination with name mobility. Domains and migration steps are governed by a parametric *accessibility relation*.

The type construct $@_w$ is intended to address the lack of expressiveness of extant session frameworks by placing domain migration information explicitly in typed interfaces. For instance, we may refine the type WStore above as follows:

$$\mathsf{WStore}_{\mathsf{sec}} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : @_{\mathsf{sec}}\, \mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

Intuitively, $\mathsf{WStore}_{\mathsf{sec}}$ decrees that communication behavior pertinent to type Payment should be preceded by a migration step to trusted domain sec, which should be directly reachable from $\mathsf{WStore}_{\mathsf{sec}}$'s current domain. Note that that migration for Payment can be specified quite precisely within the type.

Despite the expressiveness gain brought by type $@_{\mathsf{sec}}$, type $\mathsf{WStore}_{\mathsf{sec}}$ may not be flexible enough, for domain sec is "hardwired" in the interface. This is inconvenient when the secure payment domain is different from sec (e.g., when using a different encryption protocol), and is especially so when such a domain is not known in advance, which is a frequent situation. Our third technical contribution is an extension of the interpretation in § IV to dynamic domain associations (§ V). We extend our type structure with universal and existential quantification over domains. This brings flexibility and expressiveness: by interpreting these connectives simply as the communication of domain identities, session types may refer to *unknown* domains which are resolved only at runtime. Returning to our example, we may define the following type:

$$\mathsf{WStore}_{\exists} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \exists\alpha.\, @_{\alpha}\, \mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

In $\mathsf{WStore}_{\exists}$, α stands for a *domain variable*: intuitively, it represents a directly reachable payment domain, unknown *a priori* to clients of $\mathsf{WStore}_{\exists}$. As we will see, a process typed with $\mathsf{WStore}_{\exists}$ will *output* the identity w of the domain α to the client, after which actual domain migration may follow. Under some assumptions about the reachability relation, communication to domain w will be secure (see § VII).

Our fourth contribution concerns the strong guarantees for domain-aware processes which are ensured by typing (§ VI). The logical foundations of our framework ensure that well-typed processes enjoy the following guarantees: session fidelity (type preservation), global progress (lock-freedom), and termination (strong normalisation). These are same properties that well-typed processes in [6] enjoy.

The above properties related to correctness of structured communication are complemented by a property tied to the domain-awareness dimension of our framework, dubbed *domain preservation*, ensuring that process communication always respects domain-related conditions as defined by the accessibility relation. The second property, *domain parametricity*, intuitively guarantees that well-typed processes respect prescribed session types independently of the domain information provided by their environment at runtime.

We also note that our framework assumes domain unconnectedness by default: because the domains (and related accessibility relations) in which session behaviors are available are often only partially known, our framework does not fix *a priori* the domain connectivity. This fact is crucial to consistently enforce valid intra-domain interactions by typing.

In § VII we develop three different applications of our framework. They illustrate how domains in our setting may admit several useful interpretations: § VII-A discusses our motivating example in greater depth; § VII-B revisits $\lambda 5$, a Curry-Howard interpretation of intuitionistic modal logic S5 that models sequential distributed computation [7]; and § VII-C shows how to capture information flow with declassification in our system, where security levels are encoded using the accessibility relation and the declassified information is precisely specified in types of "high" processes.

In our view, the expressiveness of our typed process framework significantly improves on that of known session types disciplines. To the best of our knowledge, our framework is the first in adopting a unified view of structured communication and domain-awareness concerns via a purely logic approach.

II. Process Model: Syntax and Semantics

We introduce a variant of the synchronous π-calculus [8] with labeled choice and domain migration via communication.

Definition 2.1: Given infinite, disjoint sets Λ of *names* (x, y, z, u, v) and $\mathcal{W}$ of *domain tags* (w, w', w''), respectively, the set of *processes* (P, Q, R) is defined by

$$\begin{array}{rlll} P ::= & [x \leftrightarrow y] & \mid P \mid Q & \mid (\boldsymbol{\nu} y)P \\ \mid & x\langle y\rangle.P & \mid x(y).P & \mid !x(y).P \\ \mid & x.\mathtt{case}(P,Q) & \mid x.\mathtt{inr}; P & \mid x.\mathtt{inl}; P \\ \mid & x\langle y@w\rangle.P & \mid x(y@w).P & \mid \mathbf{0} \end{array}$$

Domain tags are present only in the migration prefixes of the last line. As we make precise in the typed setting of § IV, these constructs realize mobility, in the usual sense of the π-calculus: migration to a domain is always associated to mobility with a fresh name. For this reason, our notation for migration prefixes is close to that for input and output prefixes.

The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition) and $(\boldsymbol{\nu} y)P$ (name restriction) are standard. We then have $x\langle y\rangle.P$ (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. Constructs in the third line define a minimal labeled choice mechanism; our model handles binary choice, without loss of generality. The first two operators in the fourth line define explicit domain migration: given a domain w, $x\langle y@w\rangle.P$ denotes a process that is prepared to migrate the communication actions in P on endpoint x, to session y on w. Complementarily, process $x(y@w).P$ signals an endpoint x to move to w, 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. Following [9], we abbreviate $(\boldsymbol{\nu} y)x\langle y\rangle$ and $(\boldsymbol{\nu} y)x\langle y@w\rangle$ as $\overline{x}\langle y\rangle$ and $\overline{x}\langle y@w\rangle$, respectively.

In $(\boldsymbol{\nu} y)P$, $x(y).P$, and $x(y@w).P$ the distinguished occurrence of name y is binding with scope P. We identify processes up to consistent renaming of bound names, writing $\equiv_\alpha$ for this congruence. $P\{x/y\}$ denotes the capture-avoiding substitution of x for y in P. While *structural congruence* expresses identities on the basic structure of processes, *reduction* expresses their behavior.

Definition 2.2: Structural congruence ($P \equiv Q$) is the least congruence relation on processes such that

$$\begin{array}{c} P \mid \mathbf{0} \equiv P \qquad P \equiv_\alpha Q \Rightarrow P \equiv Q \qquad (\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0} \\ P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\ x \notin \mathit{fn}(P) \Rightarrow P \mid (\boldsymbol{\nu} x)Q \equiv (\boldsymbol{\nu} x)(P \mid Q) \\ [x \leftrightarrow y] \equiv [y \leftrightarrow x] \qquad (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P \end{array}$$

Definition 2.3: Reduction ($P \to Q$) is the binary relation on processes defined by the rules in Fig. 1.

By definition, reduction is closed under $\equiv$. It specifies the computations that a process performs on its own. To define the interactions between a process and its environment, in § VI we give a labeled transition system for well-typed processes.

For the sake of generality, reduction allows two complementary endpoints with the same name to interact, independently of the domains of the involved subjects. The type discipline in § IV will ensure, among other things, *local reductions*, thus disallowing synchronisations among distinct, possibly unreachable domains.

$$\begin{array}{c} x\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\} \\ x\langle y\rangle.Q \mid !x(z).P \to Q \mid P\{y/z\} \mid !x(z).P \\ (\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \to P\{y/x\} \quad (x \neq y) \\ Q \to Q' \Rightarrow P \mid Q \to P \mid Q' \\ P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q \\ P \equiv P' \wedge P' \to Q' \wedge Q' \equiv Q \Rightarrow P \to Q \\ x.\mathtt{inr}; P \mid x.\mathtt{case}(Q, R) \to P \mid R \\ x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) \to P \mid Q \\ x\langle y@w\rangle.P \mid x(z@w').Q \to P \mid Q\{y/z\} \end{array}$$

Fig. 1. Reduction for domain-aware processes

III. Hybridized Intuitionistic Linear Logic (HILL)

Hybrid logic is often used as an umbrella term for a class of logics that extend the expressiveness of traditional modal logic by considering modal worlds as *syntactic objects* in propositions. Here we present a particular formulation of hybrid logic via the so-called judgmental approach (cf. [10]), introducing the key connectives and their intended meaning in a linear setting.

Recall $w, w_1, \ldots$ stand for elements of $\mathcal{W}$. Let us use $\alpha, \beta, \ldots$ to range over *world variables*. We consider a generalization of intuitionistic linear logic with hybrid logic operators, where modal worlds are subject to an *accessibility relation*, tracked in a separate context Ω. Intuitively, Ω collects direct reachability hypotheses of the form $w_1 \prec w_2$, meaning that world w_2 is directly reachable from world w_1. We call this logic *hybridized intuitionistic linear logic* (HILL, in the sequel).

*Definition 3.1 (*HILL *propositions):* The syntax of propositions (A, B, C) is defined by

$$\begin{array}{rllll} A ::= & \mathbf{1} & \mid A \multimap B & \mid A \otimes B & \\ \mid & @_w A & \mid A \,\&\, B & \mid A \oplus B & \mid !A \end{array}$$

Logic connectives in HILL include multiplicative conjunction and implication ($\otimes$ and $\multimap$, respectively), the multiplicative unit $\mathbf{1}$, the additives $\&$ and $\oplus$ and the exponential $!$, all of which have their usual meanings [11]. The only hybrid logic connective that we consider is $@_w A$: it denotes that A is true at world w, *directly reachable* from the current world. Our system consists of two judgments:

$$\text{(i)}\ \ \Omega \vdash w_1 \prec w_2 \qquad \text{and} \qquad \text{(ii)}\ \ \Omega; \Gamma; \Delta \vdash A[w]$$

Judgment (i) states that w_1 can directly reach w_2 under the hypotheses in Ω. We omit Ω when it is clear from context. We write $\prec^*$ for the *reflexive transitive closure* of $\prec$, and $w_1 \not\prec^* w_2$ when $w_1 \prec^* w_2$ does not hold. Judgment (ii) states that proposition A is true at world w, under the direct reachability hypotheses Ω, *unrestricted* (or exponential) hypotheses Γ, and *linear* hypotheses Δ, where weakening and contraction principles hold for Γ but not for Δ. Note that each hypothesis in Γ and Δ is itself labeled with a specific world. When empty, Ω, Γ, Δ are denoted by '$\cdot$'.

$$\text{(whyp)}\ \frac{}{\Omega, w_1 \prec w_2 \vdash w_1 \prec w_2} \qquad \text{(id)}\ \frac{}{\Omega;\Gamma; x{:}A[w] \vdash [x\leftrightarrow z] :: z{:}A[w]}$$

$$\text{(}\mathbf{1}\text{L)}\ \frac{\Omega;\Gamma;\Delta \vdash P :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}\mathbf{1}[w_2] \vdash P :: z{:}C[w_1]}$$

$$(\multimap\text{R})\ \frac{\Omega;\Gamma;\Delta, y{:}A[w] \vdash P :: z{:}B[w]}{\Omega;\Gamma;\Delta \vdash z(y).P :: z{:}A \multimap B[w]} \qquad \text{(}\mathbf{1}\text{R)}\ \frac{}{\Omega;\Gamma;\cdot \vdash \mathbf{0} :: x{:}\mathbf{1}[w]}$$

$$(\otimes\text{R})\ \frac{\Omega;\Gamma;\Delta_1 \vdash P :: y{:}A[w] \quad \Omega;\Gamma;\Delta_2 \vdash Q :: z{:}B[w]}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash \overline{z}\langle y\rangle.(P \mid Q) :: z{:}A \otimes B[w]}$$

$$(\otimes\text{L})\ \frac{\Omega;\Gamma;\Delta, y{:}A[w_2], x{:}B[w_2] \vdash P :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}A \otimes B[w_2] \vdash x(y).P :: z{:}C[w_1]}$$

$$(\multimap\text{L})\ \frac{\Omega;\Gamma;\Delta_1 \vdash P :: y{:}A[w_2] \quad \Omega;\Gamma;\Delta_2, x{:}B[w_2] \vdash Q :: z{:}C[w_1]}{\Omega;\Gamma;\Delta_1,\Delta_2, x{:}A \multimap B[w_2] \vdash \overline{x}\langle y\rangle.(P \mid Q) :: z{:}C[w_1]}$$

$$(@\text{R})\ \frac{\Omega \vdash w_1 \prec w_2 \quad \Omega \vdash w_2 \prec^* \Delta \quad \Omega;\Gamma;\Delta \vdash P :: y{:}A[w_2]}{\Omega;\Gamma;\Delta \vdash \overline{z}\langle y@w_2\rangle.P :: z{:}@_{w_2}A[w_1]}$$

$$(@\text{L})\ \frac{\Omega, w_2 \prec w_3;\Gamma;\Delta, y{:}A[w_3] \vdash P :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}@_{w_3}A[w_2] \vdash x(y@w_3).P :: z{:}C[w_1]}$$

$$\text{(copy)}\ \frac{\Omega \vdash w_1 \prec^* w_2 \quad \Omega;\Gamma, u{:}A[w_2];\Delta, y{:}A[w_2] \vdash P :: z{:}C[w_1]}{\Omega;\Gamma, u{:}A[w_2];\Delta \vdash \overline{u}\langle y\rangle.P :: z{:}C[w_1]}$$

$$\text{(cut)}\ \frac{\begin{array}{c}\Omega \vdash w_1 \prec^* w_2 \quad \Omega \vdash w_2 \prec^* \Delta_1 \\ \Omega;\Gamma;\Delta_1 \vdash P :: x{:}A[w_2] \quad \Omega;\Gamma;\Delta_2, x{:}A[w_2] \vdash Q :: z{:}C[w_1]\end{array}}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C[w_1]}$$

$$(\text{cut}^!)\ \frac{\Omega;\Gamma;\cdot \vdash P :: x{:}A[w_1] \quad \Omega;\Gamma, u{:}A[w_1];\Delta \vdash Q :: z{:}C[w_2]}{\Omega;\Gamma;\Delta \vdash (\boldsymbol{\nu} u)(!u(x).P \mid Q) :: z{:}C[w_2]}$$

Fig. 2. HILL with proof terms in blue (omitted rules in Fig. 5)

We consider a sequent calculus presentation of HILL, made up of so-called right rules (which specify how to *prove* a particular proposition, marked with R) and left rules (which specify how to *use* a given hypothesis, marked with L). A subset of the rules of the system is given in Fig. 2; for the sake of space, remaining rules are given in the Appendix (Fig. 5). Notice that the actual logical rules are obtained by erasing the names x, y, z, u and process terms (indicated in blue). The rules essentially correspond to those for intuitionistic linear logic DILL [11], extended with world labels and context Ω.

We consider a sequent $\Omega;\Gamma;\Delta \vdash C[w_1]$ *well-formed* if $\Omega \vdash w_1 \prec^* w_2$ for every $A[w_2] \in \Delta$, which we abbreviate as $\Omega \vdash w_1 \prec^* \Delta$, meaning that all worlds mentioned in Δ are reachable (not necessarily in a single *direct* step) from w_1. No such world requirement is imposed on Γ. If an end sequent is well-formed, every sequent in its proof will also be well-formed. For labels, all the rules (read bottom-up) will preserve this invariant and only (cut), (copy), (@R) (and later (∀L) and (∃R)) require explicit reachability checks (see below). We can therefore restrict our attention to well-formed sequents. This invariant will be of paramount importance in the interpretation of § IV, since it automatically excludes the possibility of processes on unreachable domains to interact (cf. Theorem 6.6).

We briefly discuss some of the rules. Rule (⊗R) allows us to prove $A \otimes B$ at w by splitting the linear context into two parts, one used to prove $A[w]$ and the other to prove $B[w]$. The cut rule enables compositional reasoning: if we can prove C at w_1 using $A[w_2]$, are also able to prove $A[w_2]$ and that $w_1 \prec^* w_2$ (in order to maintain the invariant that all worlds tagging linear hypotheses must be reachable), then we can prove $C[w_1]$ outright. Rule (@R) states that in order for $@_{w_2}A$ to be true at w_1, we must be able to directly reach w_2 from w_1 and A must be true at w_2. Rule (@L) allows us to use an assumption of $@_{w_3}A$ at w_2 to prove C at w_1, which justifies the use of assumption A at w_3 to prove $C[w_1]$.

Following our previous remark on well-formed sequents, the only rules that appeal to accessibility are (@R), (@L), (copy), and (cut). We notice that these conditions are directly associated with varying degrees of flexibility/expressiveness in terms of provability, depending on what relationship is imposed between the world to the left and to the right of the turnstile in the left rules. Most importantly, the conditions have consequences on the metatheory of the logic, in particular, admissibility of cut (Theorem 3.3). Moreover, from the standpoint of the process interpretation of HILL in § IV, these conditions also impact on the kinds of typable processes.

It is straightforward to see that HILL orthogonally extends DILL: we may recover the latter by tagging every proposition with the same world with a reflexive accessibility relation. Moreover, the usual (and central) soundness meta-theoretic property of cut elimination holds for HILL, by relying on the following *world substitution* property:

Lemma 3.2 (World Substitution): If $\Omega \vdash w_1 \prec w_2$ and $\Omega, w_1 \prec w_2;\Gamma;\Delta \vdash A[w]$ then $\Omega;\Gamma;\Delta \vdash A[w]$.

Theorem 3.3 (Cut Admissibility): If $\Omega;\Gamma;\Delta \vdash A[w]$ then there exists a derivation of $\Omega;\Gamma;\Delta \vdash A[w]$ that does not use the rules (cut) or (cut$^!$).

The two judgment presentation enables us to consider a particular accessibility relation as a *parameter* of the framework. This allows changing accessibility relations without having to modify the entire system. To consider the simplest possible accessibility relation, the only defining rule for accessibility would be rule (whyp) in Fig. 2. To consider an accessibility relation which is an equivalence relation, then we would add reflexivity, transitivity, and symmetry rules to the judgment.

IV. HILL PROPOSITIONS AS DOMAIN-AWARE SESSION TYPES

Here we generalize the interpretation of DILL propositions as session types given in [6] to the case of HILL. As in [6], propositions are interpreted as session types, proofs as typing derivations, and proof reduction as process communication. Moreover, we interpret worlds as domains, and the hybrid connective $@_w A$ as as the type of a session that moves to a reachable domain w.

We now make these intuitions precise. Our type syntax coincides exactly with that of HILL propositions (cf. Def. 3.1). For simplicity, we do not consider base types. In our setting, types are assigned to names; a *type assignment* $x{:}A[w]$ enforces the use of name x according to session A, *in the domain* w. A *type environment* is a collection of type assignments. Besides the accessibility context Ω, our typing judgments consider two kinds of type environments, subject to different structural properties: a *linear* part Δ and an *unrestricted* part Γ. A judgment is of the form

$$\Omega; \Gamma; \Delta \vdash P :: z{:}A[w_1]$$

where name declarations in Γ are always propagated unchanged to all premises in the typing rules, while name declarations in Δ are handled multiplicatively or additively, depending on the type being defined.

Such a judgment asserts: P is ensured to safely provide the session behavior A on channel z, at a *domain* w, and using the located sessions specified in Δ (linearly) and Γ (unrestrictedly) and adhering to the reachability assumptions in Ω. Conditions on well-formed HILL sequents (described in the previous section) also have a role in judgments for processes, and they extend accordingly. Formally, judgment $\Omega; \Gamma; \Delta \vdash z{:}A[w_1]$ is well-formed if all the variables in Γ, Δ, and z are pairwise distinct, and $\Omega \vdash w_1 \prec^* \Delta$ Recall that no such world requirement is imposed on Γ. For variables, the invariant is enforced by implicit α-conversion, as usual. Since we identify a process by the channel along which it offers communication, the world invariant expresses that a process only uses sessions in reachable domains.

We always consider processes modulo structural congruence; hence, typability is closed under $\equiv$ by definition. The typing rules for the system are given in Fig. 2. While right rules specify how to *offer* a session of a given type, left rules define how to *use* a session.

We recall some key features of the interpretation in [6], which covers the non hybrid connectives in Def. 3.1. Type $A \multimap B$ denotes a session that inputs a channel of type A and proceeds as B. To offer $z{:}A \multimap B$ at domain w, we input along z a channel y that will offer A at w and proceed, now offering $z{:}B$ at w:

$$(\multimap\mathsf{R})\ \frac{\Omega; \Gamma; \Delta, y{:}A[w] \vdash P :: z{:}B[w]}{\Omega; \Gamma; \Delta \vdash z(y).P :: z{:}A \multimap B[w]}$$

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 $z{:}B$. In order to type linear composition, we compose process P offering $x{:}A[w_2]$ with Q using $x{:}A[w_2]$ to offer $z{:}C[w_1]$, provided that $w_1 \prec^* w_2$, and bind the scope of x to the two processes:

$$(\mathsf{cut})\ \frac{\begin{array}{c}\Omega \vdash w_1 \prec^* w_2 \quad \Omega \vdash w_2 \prec^* \Delta_1 \\ \Omega; \Gamma; \Delta_1 \vdash P :: x{:}A[w_2] \quad \Omega; \Gamma; \Delta_2, x{:}A[w_2] \vdash Q :: z{:}C[w_1]\end{array}}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C[w_1]}$$

Type $\mathbf{1}$ means that the session terminated, no further interaction will take place on it; names of type $\mathbf{1}$ may still be passed around as opaque values. $A \& B$ types a session channel that offers its partner a choice between an A behavior ("left" choice) and a B behavior ("right" choice). Dually, $A \oplus B$ types a session that either selects "left" and then proceeds as specified by A, or else selects "right", and then proceeds as specified by B. Type $!A$ 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.

We now detail the elements of the interpretation induced by HILL which represent enhancements over [6]. First, we interpret the logical notion of world as a form of domain—a designated site at which a session behavior is available. This interpretation is natural, and gives rise to a form of explicitly distributed session behavior that is described precisely by typing assignments—which, as explained before, relate names, session types, and domains. Typing rules obtained from the sequent calculus for HILL ensure that inter-domain interaction is only allowed between Ω-reachable domains.

Even if by virtue of the typing assignments labeled by domains we obtain a model of distributed session behavior, such a model is inherently static. To obtain more flexibility, the hybrid type operator $@_w$ enables mobility, regulated by Ω. A channel typed with $@_{w_2}A$ denotes that behavior A is available by first *moving to* domain w_2, directly accessible from the current domain. More precisely, we have:

$$(@\mathsf{R})\ \frac{\Omega \vdash w_1 \prec w_2 \quad \Omega \vdash w_2 \prec^* \Delta \quad \Omega; \Gamma; \Delta \vdash P :: y{:}A[w_2]}{\Omega; \Gamma; \Delta \vdash \overline{z}\langle y@w_2 \rangle.P :: z{:}@_{w_2}A[w_1]}$$

Hence, given domains w_1 and w_2, a process *offering* a behavior $z{:}@_{w_2}A$ at w_1 ensures: (i) behavior A is available at w_2 along a *fresh* session channel y that is emitted along z and (ii) w_2 is directly reachable from w_1. To maintain well-formedness of the sequent we also need to check that all domains in Δ are still accessible from w_2.

Dually, *using* a service $x{:}@_{w_3}A[w_2]$ entails receiving a channel y that will offer the behavior A at domain w_3 (and also allowing the usage of the fact that $w_2 \prec w_3$):

$$(@\mathsf{L})\ \frac{\Omega, w_2 \prec w_3; \Gamma; \Delta, y{:}A[w_3] \vdash P :: z{:}C[w_1]}{\Omega; \Gamma; \Delta, x{:}@_{w_3}A[w_2] \vdash x(y@w_3).P :: z{:}C[w_1]}$$

This way, in our interpretation the type operator $@_w$ 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. However, processes themselves do not necessarily need to move across domains in the sense of mobile code, since domains are more abstract than physical locations. See §VIII for discussions of related work.

Notice how the interpretation for the non-hybrid connectives is as in [6]. We note the distinction between direct reachability $\prec$ in the hybrid connective rules and reachability $\prec^*$, potentially requiring multiple accessibility "hops", in the (copy) and (cut) rules, which enforces the reachability invariant. This pertains to the fact that we localize domain information at the level of specifications, which is made explicit when we define how to *offer* a service: to specify a service requiring domain

$$(\forall\mathsf{R})\ \frac{\Omega, w_1 \prec \alpha; \Gamma; \Delta \vdash P :: z{:}A[w_1]}{\Omega; \Gamma; \Delta \vdash z(\alpha).P :: z{:}\forall\alpha.A[w_1]}$$

$$(\forall\mathsf{L})\ \frac{\Omega \vdash w_2 \prec w_3 \quad \Omega; \Gamma; \Delta, x{:}A\{w_3/\alpha\}[w_2] \vdash Q :: z{:}C[w_1]}{\Omega; \Gamma; \Delta, x{:}\forall\alpha.A[w_2] \vdash x\langle w_3\rangle.Q :: z{:}C[w_1]}$$

$$(\exists\mathsf{R})\ \frac{\Omega \vdash w_1 \prec w_2 \quad \Omega; \Gamma; \Delta \vdash P :: z{:}A\{w_2/\alpha\}[w_1]}{\Omega; \Gamma; \Delta \vdash z\langle w_2\rangle.P :: z{:}\exists\alpha.A[w_1]}$$

$$(\exists\mathsf{L})\ \frac{\Omega, w_2 \prec \alpha; \Gamma; \Delta, x{:}A[w_2] \vdash Q :: z{:}C[w_1]}{\Omega; \Gamma; \Delta, x{:}\exists\alpha.A[w_2] \vdash x(\alpha).Q :: z{:}C[w_1]}$$

Fig. 3. HILL with quantification over worlds: Additional rules.

migration, we detail each migration step explicitly in terms of Ω-reachability. For instance, we may want to distinguish specifications of services that migrate from w_1 to w_2 to w_3 from those that migrate directly from w_1 to w_3. However, the usage of such services from the point of view of a client located at, say w_1, should only be constrained by being able to reach w_3 in *some* way (i.e., using $\prec^*$), not by the particular migration steps the service takes. Of course, if we postulate for a class of applications that $\prec$ is reflexive and transitive, the two notions collapse into one.

Why is it necessary to consider reflexivity and transitivity at all? First note that the identity rule (id) forwards between two sessions in the same domain. Therefore, the current domain should be accessible. Second, note that the (cut) rule, read as composition, requires domains in Δ_1, the hypotheses of the first premise, to be accessible from w_1, the domain of the second premise. This follows from transitivity of $\prec^*$ using the intermediary w_2 but might otherwise be compromised.

V. Dynamic Domain Associations via Quantification over Worlds

The system given thus far allows for dynamic associations between domains and typing assignments through the @ operator. However, the domains themselves are fixed in the sense that they are all known *a priori* and a type must always mention a concrete (reachable) domain. To obtain extra flexibility and expressiveness, we may extend our framework with two other hybrid connectives, namely with a form of universal and existential quantification over worlds.

At the process level, domain-quantified session types introduce domains as 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. The typing rules for domain-quantified session types are given in Fig. 3 (the associated logical rules can be recovered by erasing the names and process terms in blue). We account for the hybrid quantifiers by extending the process syntax with domain output and input prefixes $z\langle w\rangle.P$ and $z(\alpha).P$, respectively (where w is a domain and α a domain variable). Rule ($\forall$R) says that a process seeking to offer $\forall\alpha.A[w_1]$ denotes a service that while located at domain w_1, may refer to any domain directly reachable from w_1 in its service specification (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 α. Dually, rule ($\forall$L) says that a process that interacts with a service of type $x{:}\forall\alpha.A[w_2]$ must make concrete the domain that is directly reachable from w_2 it wishes to use, which is achieved by the appropriate output action. Rules ($\exists$L) and ($\exists$R) for the existential quantifier have a dual reading.

The interpretation of universal and existential quantification as domain communication thus requires the reduction rule (with the expected notion of capture-avoiding world substitution):

$$x\langle w\rangle.P \mid x(\alpha).Q \to P \mid Q\{^{w}\!/\!_{\alpha}\}$$

This is justified logically by the principal cut reductions of the connectives, appealing to a domain parametricity theorem and Lemma 3.2 in order to account for domain variables.

Theorem 5.1 (Domain Parametricity):
Let α be a domain variable, and w_1, w, w' be domains (constants or variables). If $\Omega, w_1 \prec \alpha; \Gamma; \Delta \vdash P :: z{:}A[w']$ then $\Omega, w_1 \prec w; \Gamma'; \Delta' \vdash P\{w/\alpha\} :: z{:}A\{w/\alpha\}[w'\{w/\alpha\}]$, where $\Gamma' = \Gamma\{w/\alpha\}$ and $\Delta' = \Delta\{w/\alpha\}$.

VI. Technical Results

We now state the main results for our typed model, namely type safety in the form of type preservation (Theorem 6.3) and global progress (Theorem 6.4). These results directly ensure session fidelity and deadlock-freedom. Also, typing ensures that processes are terminating, i.e., they do not exhibit infinite reduction paths (Theorem 6.5). Moreover, as a property specific to domain-aware processes, we show that well-typed processes satisfy *domain preservation*, which says that processes always respect their associated domain accessibility conditions (Theorem 6.6).

Some Preliminaries

1) 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 π-calculus [8] 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 λ. Transition labels are defined below:

$$\begin{array}{rcl} \lambda & ::= & \tau \mid x(y) \mid x(w) \mid x.\mathsf{inl} \mid x.\mathsf{inr} \mid x.y@w \\ & \mid & \overline{x\,y} \mid \overline{x\langle y\rangle} \mid \overline{x\,w} \mid \overline{x.\mathsf{inl}} \mid \overline{x.\mathsf{inr}} \mid \overline{x.y@w} \end{array}$$

Actions are name input $x(y)$, domain input $x(w)$, the offers $x.\mathsf{inl}$ and $x.\mathsf{inr}$, migration $x.y@w$ and their matching co-actions, respectively the output $\overline{x\,y}$ and bound output $\overline{x\langle y\rangle}$ actions, the domain output $\overline{x\,w}$, the left/ right selections $\overline{x.\mathsf{inl}}$ and $\overline{x.\mathsf{inr}}$, and domain migration $\overline{x.y@w}$. Both the bound output $\overline{x\langle y\rangle}$ and migration action $\overline{x.y@w}$ denote extrusion of a fresh name y along x. Internal action is denoted by

$$\textsf{(id)}\ (\boldsymbol{\nu}x)([x\leftrightarrow y] \mid P) \xrightarrow{\tau} P\{y/x\}$$

$$\textsf{(n.out)}\ x\langle y\rangle.P \xrightarrow{\overline{x\,y}} P \qquad \textsf{(n.in)}\ x(y).P \xrightarrow{x(z)} P\{z/y\}$$

$$\textsf{(d.out)}\ x\langle w\rangle.P \xrightarrow{\overline{x\,w}} P \qquad \textsf{(d.in)}\ x(\alpha).P \xrightarrow{x(w)} P\{w/\alpha\}$$

$$\textsf{(move)}\ x\langle y@w\rangle.P \xrightarrow{\overline{x.y@w}} (\boldsymbol{\nu}y)P$$

$$\textsf{(move')}\ x(z@w).P \xrightarrow{x.y@w} P\{y/z\}$$

$$\textsf{(par)}\ \frac{P \xrightarrow{\lambda} Q}{P \mid R \xrightarrow{\lambda} Q \mid R} \qquad \textsf{(com)}\ \frac{P \xrightarrow{\overline{\lambda}} P' \quad Q \xrightarrow{\lambda} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\textsf{(res)}\ \frac{P \xrightarrow{\lambda} Q}{(\boldsymbol{\nu}y)P \xrightarrow{\lambda} (\boldsymbol{\nu}y)Q} \qquad \textsf{(open)}\ \frac{P \xrightarrow{\overline{x\,y}} Q}{(\boldsymbol{\nu}y)P \xrightarrow{\overline{x\langle y\rangle}} Q}$$

$$\textsf{(close)}\ \frac{P \xrightarrow{\overline{x\langle y\rangle}} P' \quad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}y)(P' \mid Q')}$$

$$\textsf{(rep)}\ !x(y).P \xrightarrow{x(z)} P\{z/y\} \mid !x(y).P$$

$$\textsf{(l.out)}\ x.\mathtt{inl};P \xrightarrow{\overline{x.\mathsf{inl}}} P \qquad \textsf{(r.out)}\ x.\mathtt{inr};P \xrightarrow{\overline{x.\mathsf{inr}}} P$$

$$\textsf{(l.in)}\ x.\mathsf{case}(P,Q) \xrightarrow{x.\mathsf{inl}} P \qquad \textsf{(r.in)}\ x.\mathsf{case}(P,Q) \xrightarrow{x.\mathsf{inr}} Q$$

Fig. 4. Labeled Transition System.

τ. In general, an action requires a matching co-action in the environment to enable progress.

Definition 6.1 (Labeled Transition System): The relation *labeled transition* ($P \xrightarrow{\lambda} Q$) is defined by the rules in Fig. 4, subject to the side conditions: in rule (res), we require $y \notin fn(\lambda)$; in rule (par), we require $bn(\lambda) \cap fn(R) = \emptyset$; in rule (close), we require $y \notin fn(Q)$. We omit the symmetric versions of rules (par), (com), and (close).

We write $subj(\lambda)$ for the subject of the action λ, 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 ρ_1, ρ_2 and $\Longrightarrow$ for the reflexive, transitive closure of $\xrightarrow{\tau}$. Notation $\overset{\lambda}{\Longrightarrow}$ stands for $\Longrightarrow\xrightarrow{\lambda}\Longrightarrow$ (given $\lambda \neq \tau$) and $\overset{\tau}{\Longrightarrow}$ stands for $\Longrightarrow$. We recall basic facts about reduction, structural congruence, and labeled transition: closure of labeled transitions under structural congruence, and coincidence of τ-labeled transition and reduction [8]: (1) if $P \equiv\xrightarrow{\lambda} Q$ then $P \xrightarrow{\lambda}\equiv Q$; (2) $P \to Q$ iff $P \xrightarrow{\tau}\equiv Q$.

A. Session Fidelity, Global Progress, and Termination

*a) **Type Preservation**:* Following [6], our proof of type preservation relies on a simulation between reductions in the session-typed π-calculus and proof reductions from logic. Crucially, there are a series of reduction lemmas that relate process actions with parallel composition through the (cut) rule. For instance, for the universal quantifier over worlds introduced in the previous section, the lemma is given below.

Lemma 6.2 (Reduction Lemma - $\forall$): Assume
(a) $\Omega; \Gamma; \Delta_1 \vdash P :: x{:}\forall\alpha.A[w_2]$ with $P \xrightarrow{x(w_3)} P'$ and
(b) $\Omega; \Gamma; \Delta_2, x{:}\forall\alpha.A[w_2] \vdash Q :: z{:}C[w_1]$ with $Q \xrightarrow{\overline{x w_3}} Q'$.
Then: $\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu}x)(P' \mid Q') :: z{:}C[w_1]$

By appealing to such reduction lemmas, we can then establish:

Theorem 6.3 (Type Preservation): If $\Omega; \Gamma; \Delta \vdash P :: z{:}A[w]$ and $P \to Q$ then $\Omega; \Gamma; \Delta \vdash Q :: z{:}A[w]$.

*b) **Progress**:* The proof of global progress also follows the lines of [6]: it relies on a series of inversion lemmas and 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 $P \equiv (\boldsymbol{\nu}\tilde{n})(\pi.Q \mid R)$, for some process R, sequence of names $\tilde{n}$ and a non-replicated guarded process $\pi.Q$.

Theorem 6.4 (Global Progress): If $\Omega; \cdot; \cdot \vdash P :: x{:}\mathbf{1}[w]$ and $live(P)$ then $\exists Q$ s.t. $P \to Q$.

Note that the condition on the contexts and on the type for name x in Theorem 6.4 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 ($\mathbf{1}$R).

*c) **Termination**:* Termination 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 $P \Downarrow$, if there is no infinite reduction path from P.

Theorem 6.5 (Termination): If $\Omega; \Gamma; \Delta \vdash P :: x{:}A[w]$ then $P \Downarrow$.

Proof (Sketch): By adapting the *linear* logical relations given in [12], [13]. For the system in § IV (without quantifiers), the logical relations correspond to those in [12], extended to carry over Ω. For the system with quantifiers given in § V, the logical relations resemble more those proposed for polymorphic session types in [13], but with the crucial observation that no impredicativity concerns are involved. ■

B. 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. We make this observation precise with the following theorem, showing that all the rules of our typing system preserve well-formedness.

Theorem 6.6: Let $\mathcal{E}$ be a derivation of $\Omega; \Gamma; \Delta \vdash P :: z{:}A[w]$. If $\Omega; \Gamma; \Delta \vdash P :: z{:}A[w]$ is well-formed then every sub-derivation in $\mathcal{E}$ well-formed.

Combining Theorems 6.3 and 6.6, we can show that processes will only be connected, along linear session channels, to processes located in reachable domains throughout their computation. While persistent session channels with unreachable domains may appear in the context Γ throughout the typing derivation, such channels can never be used and thus can not appear in a well-typed process due the restriction on the (copy) rule.

VII. Examples

A. Revisiting the E-Commerce Example

We now revisit the motivating example of the Introduction and make it precise by applying the techniques we have developed. Recall the refined web store session type:

$$\begin{array}{l}\mathsf{WStore}_{\mathtt{sec}} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : @_{\mathtt{sec}}\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\} \\ \mathsf{Payment} \triangleq \mathsf{ShipAddress} \multimap \mathsf{CCNumber} \multimap \mathsf{Receipt} \otimes \mathbf{1}\end{array}$$

The $@_w$ type constructor allows us to express a very precise form of coordinated domain migration. Consider the $\mathsf{WStore}_{\mathtt{sec}}$ service interacting with a client process along channel x, each in their own (reachable) domains, $\mathtt{c}$ and $\mathtt{ws}$, 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:

$$\mathtt{c} \prec \mathtt{ws}; \cdot; x{:}@_{\mathtt{sec}}\mathsf{Payment}[\mathtt{ws}] \vdash \mathit{Client} :: z{:}@_{\mathtt{sec}}\mathbf{1}[\mathtt{c}]$$

At this point, it is *impossible* for a (typed) client to interact with the behavior that is protected by the trusted domain $\mathtt{sec}$, since it is not the case that $\mathtt{c} \prec^* \mathtt{sec}$. 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 $\mathtt{c} \prec \mathtt{ws}; \cdot; \mathsf{Payment}[\mathtt{sec}] \vdash \mathit{Client} :: z{:}@_{\mathtt{sec}}\mathbf{1}[\mathtt{c}]$ exists (Theorem 6.6). The client can only communicate in the secure domain *after* the web store service has migrated accordingly:

$$\begin{array}{l}\mathtt{c} \prec \mathtt{ws}, \mathtt{ws} \prec \mathtt{sec}; \cdot; x'{:}\mathsf{Payment}[\mathtt{sec}] \vdash \mathit{Client}' :: z'{:}\mathbf{1}[\mathtt{sec}] \\ \text{where} \quad \mathit{Client} \triangleq x(x'@\mathtt{sec}).\overline{z}\langle z'@\mathtt{sec}\rangle.\mathit{Client}'\end{array}$$

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.

$$\mathsf{WStore}_{\exists} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \exists\alpha.\, @_{\alpha}\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

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

$$\mathsf{WStore}_{\forall} \triangleq \mathsf{addCart} \multimap \&\{\mathtt{buy} : \forall\alpha.\, @_{\alpha}\,\mathsf{Payment}, \mathtt{quit} : \mathbf{1}\}$$

B. Spatial Distribution as in $\lambda 5$

Murphy et al. [7] have proposed a Curry-Howard interpretation of the intuitionistic modal logic S5 [14] 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 [15] later generalized this to hybrid logic, an idea also present in [16], 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 *action at a distance* without an explicit communication visible in the elimination rule for disjunction.

The present system not only generalizes $\lambda 5$ 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 $\lambda 5$, we will not attempt here to give a full, computationally adequate interpretation of $\lambda 5$ in `HILL` (which would generalize [17]), but instead explain the spatially distributed computational interpretation of `HILL` directly.

- $c : \Box A$. Channel c offering A can be used in any domain.
- $c : \Diamond A$. Channel c is offering A in some (hidden) domain.

These are mapped into `HILL` (choosing a fresh α each time) with

$$\begin{array}{rcl}\Box A & = & \forall\alpha.@_{\alpha}A \\ \Diamond A & = & \exists\alpha.@_{\alpha}A\end{array}$$

A process $P :: c{:}\Box A[w_1]$ 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 characteristic axioms of S5, keeping in mind that for this application accessibility is reflexive, transitive, and symmetric (we make no distinction between direct reachability or reachability requiring multiple hops).

- $\mathsf{K}_{\Diamond} :: z{:}\Box(A \multimap B) \multimap \Diamond A \multimap \Diamond B[w_0]$.
  Here, $\Diamond A$ is offered along some c_1 at some world w_1 reachable from w_0. Move the offer of $\Box(A \multimap B)[w_0]$ to w_1 as c_2 and send it c_1 to obtain $c_2 : B[w_1]$. We abstract this as $\Diamond B[w_0]$, which is possible since $w_0 \prec w_1$. Intuitively, this axiom captures the fact that a session transformer $A \multimap B$ that can be used in *any* domain may be combined with a session A offered in *some* domain in order to produce a session behavior B, itself in some hidden domain.

  $$\begin{array}{rcl}\mathsf{K}_{\Diamond} & \triangleq & z(x).z(y).y(w_1).y(c_1@w_1).x\langle w_1\rangle.x(c_2@w_1). \\ & & \overline{c_2}\langle v\rangle.([c_1 \leftrightarrow v] \mid z\langle w_1\rangle.\overline{z}\langle c_3@w_1\rangle.[c_2 \leftrightarrow c_3])\end{array}$$

- $\mathsf{B} :: z{:}\Diamond\Box A \multimap \Box A[w_0]$.
  Given a session x that offers $\Box A$ at some unknown domain w_1, itself a behavior A that may be used in any domain, we offer $\Box A$ at w_0 by offering A along c_3 at some w_2 reachable from w_0. By symmetry, $w_1 \prec w_0$, and therefore by transitivity $w_1 \prec w_2$ for any $w_0 \prec w_2$, which means we can move the offer of A to any w_2. Intuitively, this axiom captures the fact that, when every domain is connected to every other domain, session behavior that can be used anywhere that is itself in a reachable but hidden domain amounts to behavior that may be used anywhere outright.

$$\begin{aligned}\mathsf{B} \triangleq\ & z(x).x(w_1).x(c_1@w_1).z(w_2).c_1\langle w_2\rangle.\\ & c_1(c_2@w_2).\overline{z}\langle c_3@w_2\rangle.[c_2 \leftrightarrow c_3]\end{aligned}$$

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

C. Information Flow with Declassification

For simplicity, we assume we have two domains H (for high security computations) and L (for low security computation), although this can easily be generalized to a lattice of security levels. We postulate $L \prec H$, but $H \not\prec L$.

By well-formedness of sequents, which is a precondition for typability, we have that in $\Omega ; \Gamma ; \Delta \vdash P :: x : A[L]$, channels in Δ can be either high ($y : B[H]$) or low ($y : B[L]$). That is, a low security process can be in touch with one or more high security processes.

Conversely, if $\Omega ; \Gamma ; \Delta \vdash P :: x : A[H]$ then all channels in Δ must be high ($y : B[H]$). This means a high security process cannot *use* a low security channel. It can therefore not communicate any information to a low security channel except through x itself (the client may be of low security, as explained in the previous paragraph).

This means that the information leaked from a high security offering process to a low security client is precisely specified by the type A. For example,

$$A \triangleq \mathsf{Password} \multimap \oplus\{\mathsf{Auth} : \mathbf{1}, \mathsf{NotAuth} : \mathbf{1}\}$$

means the high security process would receive a password and return a bit (Auth or NotAuth) indicating if the password was correct. A compliant process cannot leak the password, since it cannot be connected to any other low security process at all. In other words, the type indicates that the correctness of the password is declassified, while the password itself remains secure.

If the type of information communicated is the unit type, then we have traditional noninterference. If the $\mathbf{1}L$ rule is silent (corresponds to no process action), then this guarantees termination-sensitive noninterference: the client cannot even tell if or when the high security process terminated. If the $\mathbf{1}L$ rule is matched with a $\mathbf{1}R$ action (as in [18]), then this enforces termination-insensitive noninterference. This is, of course, assuming we have recursive types that permit us to write nonterminating processes.

VIII. Related Work

We do not know of other typed process frameworks in which issues of structured communications and domain-awareness are jointly addressed, and in which migration relies on explicit (and possibly dynamic) domain-related information.

Much previous research developed mobile process calculi enriched with dedicated constructs meant to represent distributed behavior. A salient representative is the Ambient calculus [19], in which processes may move across *ambients*—abstractions of administrative domains. Such ambients have a particular interpretation as named locations and appear at the level of processes. In particular, by ambient nesting, processes fully describe domain hierarchies. We find two major differences between Ambient-like calculi and our process model: first, while domains in our model may be read as locations, they also admit alternative readings (e.g., security levels, as in our information flow example); second, processes in our framework refer only to the domains that directly pertain their actions. In particular, unlike in Ambients, processes in our setting do not define full domain hierarchies; such a hierarchy is meant to be defined/governed externally and so processes may only have a partial view of it.

At the level of types, we find also substantial differences, for Ambient-like calculi require sophisticated type systems to control interferences in ambient movement. In type systems such as, e.g., those in [20], [21], the explicit nested hierarchies that define ambient mobility are partitioned into groups or domains so as to statically specify communication and security properties. This research, however, does not cover issues of structured communications, which is central in our work. Garralda et al. [22] integrate binary session types into an Ambient calculus variant. Rather than domain-awareness concerns, the focus of the type system is on so-called *session safety*—roughly, ensuring that intra-session communications are not interrupted by ambient mobility steps. Another representative typed process model is the *distributed π-calculus* (DPI) [23], which extends the π-calculus with flat locations, local communication, and process migration. Typed analyses for DPI [23] address access control to sites, but do not cover compliance to structured interaction protocols.

Extensions of session types with access control and information flow analyses have been proposed in [24], [25]. There are high-level similarities between those frameworks and the ideas illustrated in the example in § VII-C. However, the typed analyses given in [24] are in the more general context of multiparty sessions, while our framework is defined in the binary setting. As synthesis of the underlying ideas is an interesting item of future work.

About a decade ago, there was significant interest on how modal logic could inform the design of *sequential* programming languages for distributed computation [16], [7], [15], as mentioned in § VII-B. The present work is in part inspired by this line of research, but goes significantly beyond it by adding concurrency (requiring linear logic) and generalizing the accessibility relation rather than forcing mutual interaccessibility of worlds. The latter generalization supports additional applications, such as information flow with declassification.

On the logical side, we owe much to Simpson's seminal work [14], which provides a general account of intuitionistic modal logic by developing a labelled calculus, parametric with respect to an accessibility relation. He does not explicitly impose a relationship between the worlds on the left and the right, permitting non-local rule applications that we avoid. Simpson also does not consider hybrid connectives or linearity. Braüner and de Paiva [5] develop an intuitionistic hybrid logic, not using a labelled system, instead developing a Kripke semantics in tandem with a natural deduction system. The checks present in our left rules are similar to those in constructive provability logic (**CPL***) [26], which requires reachability via irreflexive transitivity.

Related to the mix of linearity and hybrid logic in HILL is work by Chaudhuri and Despeyroux [27]. Their logic is designed to reason about stochastic systems via proof search in a focused calculus (without a proof term assignment), with applications in the modelling of biological phenomena, where it is convenient to have accessibility be an equivalence relation.

IX. Concluding Remarks

We have proposed a Curry-Howard interpretation of hybridized intuitionistic linear logic (HILL) as domain-aware session types. Our development generalizes the interpretation put forward in [6], leading to a session type discipline with enhanced expressiveness and strong correctness properties for well-typed processes. Even if processes typeable in the system in [6] are also typeable in our domain-aware framework (by localizing each judgment at the same world), it is worth stressing that domain information in the system given here effectively rules out certain processes—notably, it disallows communication between unreachable worlds. Enforcing these constraints by typing is beyond the reach of previous works.

Through a parametric accessibility relation, our framework provides flexible reasoning about domain structures. By relying on an extension of HILL with quantifiers over worlds, we have shown how our interpretation can account for challenging scenarios in which domain information can be only determined at runtime. The logical foundations of our framework allowed us to improve on existing session type theories by introducing a new dimension of reasoning over scenarios of structured communications, based on distributed behavior defined primarily by both processes and types.

In future work, we plan to incorporate the present system in an ongoing implementation effort, combining functional and concurrent computation [18]. One interesting point will be dynamic enforcement of session types and domain boundaries in the presence of untrusted processes. A potentially useful extension of the framework in this regard is to map the computational content of determining domain reachability to some form of authentication. Another item of future work will be to explicitly relate our system to various uses of labels in information flow control (see Montagu et al. [28] for a systematic account).

References

[1] K. Honda, "Types for dynamic interaction," in *CONCUR*, ser. LNCS, vol. 715. Springer, 1993, pp. 509–523.

[2] K. Honda, V. T. Vasconcelos, and M. Kubo, "Language primitives and type discipline for structured communication-based programming," in *ESOP'98*, ser. LNCS. Springer, 1998.

[3] M. Dezani-Ciancaglini and U. de'Liguoro, "Sessions and session types: An overview," in *WS-FM 2009*, ser. LNCS, vol. 6194. Springer, 2010, pp. 1–28.

[4] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, part I/II," *Inf. Comput.*, vol. 100, no. 1, pp. 1–77, 1992.

[5] T. Braüner and V. de Paiva, "Intuitionistic hybrid logic," *J. of App. Log.*, vol. 4, pp. 231–255, 2006.

[6] L. Caires and F. Pfenning, "Session types as intuitionistic linear propositions," in *CONCUR*, ser. LNCS, vol. 6269. Springer, 2010, pp. 222–236.

[7] T. Murphy, K. Crary, R. Harper, and F. Pfenning, "A symmetric modal lambda calculus for distributed computing," in *LICS*. IEEE Computer Society, 2004, pp. 286–295.

[8] D. Sangiorgi and D. Walker, *The π-calculus: A Theory of Mobile Processes*. New York, NY, USA: Cambridge University Press, 2001.

[9] D. Sangiorgi, "pi-calculus, internal mobility, and agent-passing calculi," *Theor. Comput. Sci.*, vol. 167, no. 1&2, pp. 235–274, 1996.

[10] B.-Y. E. Chang, K. Chaudhuri, and F. Pfenning, "A Judgmental Analysis of Linear Logic," Carnegie Mellon University, Tech. Rep. CMU-CS-03-131R, 2003.

[11] A. Barber and G. Plotkin, "Dual Intuitionistic Linear Logic," Univ. of Edinburgh, Tech. Rep. LFCS-96-347, 1997.

[12] J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho, "Linear logical relations for session-based concurrency," in *ESOP*, ser. LNCS, vol. 7211. Springer, 2012, pp. 539–558.

[13] L. Caires, J. A. Pérez, F. Pfenning, and B. Toninho, "Behavioral polymorphism and parametricity in session-based communication," in *ESOP*, ser. LNCS, vol. 7792. Springer, 2013.

[14] A. Simpson, "The proof theory and semantics of intuitionistic modal logic," Ph.D. dissertation, University of Edinburgh, 1994.

[15] T. M. VII, "Modal types for mobile code," Ph.D. dissertation, Carnegie Mellon University, 2008.

[16] L. Jia and D. Walker, "Modal proofs as distributed programs (extended abstract)," in *ESOP*, ser. LNCS, vol. 2986. Springer, 2004, pp. 219–233.

[17] B. Toninho, L. Caires, and F. Pfenning, "Functions as session-typed processes," in *FoSSaCS*, ser. LNCS, L. Birkedal, Ed., vol. 7213. Springer, 2012, pp. 346–360.

[18] ——, "Higher-order processes, functions, and sessions: A monadic integration," in *ESOP*, ser. LNCS, vol. 7792, 2013, pp. 350–369.

[19] L. Cardelli and A. D. Gordon, "Mobile ambients," *Theor. Comput. Sci.*, vol. 240, no. 1, pp. 177–213, 2000.

[20] L. Cardelli, G. Ghelli, and A. D. Gordon, "Types for the ambient calculus," *Inf. Comput.*, vol. 177, no. 2, pp. 160–194, 2002.

[21] M. Bugliesi and G. Castagna, "Behavioural typing for safe ambients," *Comput. Lang.*, vol. 28, no. 1, pp. 61–99, 2002.

[22] P. Garralda, A. B. Compagnoni, and M. Dezani-Ciancaglini, "Bass: boxed ambients with safe sessions," in *PPDP*, A. Bossi and M. J. Maher, Eds. ACM, 2006, pp. 61–72.

[23] M. Hennessy and J. Riely, "Resource access control in systems of mobile agents," *Inf. Comput.*, vol. 173, no. 1, pp. 82–120, 2002.

[24] S. Capecchi, I. Castellani, M. Dezani-Ciancaglini, and T. Rezk, "Session types for access and information flow control," in *CONCUR*, ser. LNCS, vol. 6269. Springer, 2010, pp. 237–252.

[25] S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini, "Information flow safety in multiparty sessions," in *EXPRESS*, ser. EPTCS, B. Luttik and F. Valencia, Eds., vol. 64, 2011, pp. 16–30.

[26] R. J. Simmons and B. Toninho, "Constructive provability logic," *CoRR*, vol. abs/1205.6402, 2012.

[27] K. Chaudhuri and J. Despeyroux, "A Hybrid Linear Logic for Constrained Transition Systems with Applications to Molecular Biology," INRIA, Research Report, Dec. 2010. [Online]. Available: http://hal.inria.fr/inria-00402942

[28] B. Montagu, B. C. Pierce, and R. Pollack, "A theory of information-flow labels," in *Proceedings of the 26th Computer Security Foundations Symposium*. IEEE, Jun. 2013, pp. 3–17.

Appendix

A. *Additional Rules for* HILL *(Omitted from Fig. 2)*

$$(\&\mathsf{R})\ \frac{\Omega;\Gamma;\Delta\vdash P_1 :: x{:}A[w] \quad \Omega;\Gamma;\Delta\vdash P_2 :: x{:}B[w]}{\Omega;\Gamma;\Delta\vdash z.\texttt{case}(P_1,P_2) :: z{:}A\ \&\ B[w]}$$

$$(\&\mathsf{L}_1)\ \frac{\Omega;\Gamma;\Delta, x{:}A[w_2]\vdash Q :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}A\ \&\ B[w_2]\vdash x.\texttt{inl}; Q :: z{:}C[w_1]}$$

$$(\&\mathsf{L}_2)\ \frac{\Omega;\Gamma;\Delta, x{:}B[w_2]\vdash Q :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}A\ \&\ B[w_2]\vdash x.\texttt{inr}; Q :: z{:}C[w_1]}$$

$$(\oplus\mathsf{R}_1)\ \frac{\Omega;\Gamma;\Delta\vdash P :: z{:}A[w]}{\Omega;\Gamma;\Delta\vdash z.\texttt{inl}; P :: z{:}A\oplus B[w]}$$

$$(\oplus\mathsf{R}_2)\ \frac{\Omega;\Gamma;\Delta\vdash P :: z{:}B[w]}{\Omega;\Gamma;\Delta\vdash z.\texttt{inr}; P :: z{:}A\oplus B[w]}$$

$$(\oplus\mathsf{L})\ \frac{\begin{array}{c}\Omega;\Gamma;\Delta, x{:}A[w_2]\vdash Q_1 :: z{:}C[w_1]\\ \Omega;\Gamma;\Delta, x{:}B[w_2]\vdash Q_2 :: z{:}C[w_1]\end{array}}{\Omega;\Gamma;\Delta, x{:}A\oplus B[w_2]\vdash x.\texttt{case}(Q_1,Q_2) :: z{:}C[w_1]}$$

$$(!\mathsf{L})\ \frac{\Omega;\Gamma, u{:}A[w_2];\Delta\vdash P :: z{:}C[w_1]}{\Omega;\Gamma;\Delta, x{:}!A[w_2]\vdash x(u).P :: z{:}C[w_1]}$$

$$(!\mathsf{R})\ \frac{\Omega;\Gamma;\cdot\vdash Q :: y{:}A[w]}{\Omega;\Gamma;\cdot\vdash \overline{x}\langle u\rangle.!u(y).Q :: x{:}!A[w]}$$

Fig. 5. Hybridized Intuitionistic Linear Logic HILL, with proof terms: Rules for additives and exponential