Monitoring Networks through Multiparty Session Types

Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Nobuko Yoshida, Kohei Honda

Monitoring Networks through Multiparty Session Types *

Laura Bocchi[1], Tzu-Chun Chen[2], Romain Demangeon[2], Kohei Honda[2], and Nobuko Yoshida[3]

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

Abstract. In large-scale distributed infrastructures, applications are realised through communications among distributed components. The need for methods for assuring safe interactions in such environments is recognized, however the existing frameworks, relying on centralised verification or restricted specification methods, have limited applicability. This paper proposes a new theory of *monitored* π-calculus with dynamic usage of *multiparty session types* (MPST), offering a rigorous foundation for safety assurance of distributed components which asynchronously communicate through multiparty sessions. Our theory establishes a framework for semantically precise decentralised run-time enforcement and provides reasoning principles over monitored distributed applications, which complement existing static analysis techniques. We introduce asynchrony through the means of explicit routers and global queues, and propose novel equivalences between networks, that capture the notion of interface equivalence, i.e. equating networks offering the same services to a user. We illustrate our static-dynamic analysis system with an ATM protocol as a running example and justify our theory with results: satisfaction equivalence, local/global safety and transparency, and session fidelity.

1 Introduction

One of the main engineering challenges for distributed systems is the comprehensive verification of distributed software without relying on ad-hoc and expensive testing techniques. Multiparty session types (MPST) is a typing discipline for communication programming, originally developed in the π-calculus [15, 1, 3, 11, 12, 7] towards tackling this challenge. The idea is that applications are built starting from units of design called sessions. Each type of session, involving multiple roles, is first modelled from a global perspective (*global type*) and then projected onto *local types*, one for each role involved. As a verification method, the existing MPST systems focus on static type checking of endpoint processes against local types. The standard properties enjoyed by well-typed processes are

* This work has been partially sponsored by the project Leverhulme Trust Award Tracing Networks, Ocean Observatories Initiative and EPSRC EP/K011715/1, EP/G015635/1 and EP/G015481/1.

communication safety (all processes conform to globally agreed communication protocols) and freedom from deadlocks.

The direct application of the theoretical MPST techniques to the current practice, however, presents a few obstacles. Firstly, the existing type systems are targeted at calculi with first class primitives for linear communication channels and communication-oriented control flow; the majority of mainstream engineering languages would need to be extended in this sense to be suitable for syntactic session type checking. Unfortunately, it is not always straightforward to add these features to the specific host languages (e.g. linear resource typing for a very liberal language like C). Furthermore, the executable processes in a distributed system may be implemented in different languages. Secondly, for domains where dynamically typed or untyped languages are popular (e.g., Web programming), or in multi-organizational scenarios, the introduction of static typing infrastructure to support MPST may not be realistic.

This paper proposes a theoretical system addressing the above issues by enabling both static *and* dynamic verification of communicating processes. The aim is to capture the decentralised nature of distributed application development, providing better support for heterogeneous distributed systems by allowing components to be independently implemented, using different languages, libraries and programming techniques, as well as being independently verified, either statically or dynamically, while retaining the strong global safety properties of statically verified homogeneous systems.

This work is motivated in part by our ongoing collaboration with the Ocean Observatories Initiative (OOI) [17], a project to establish cyberinfrastructure for the delivery, management and analysis of scientific data from a large network of ocean sensor systems. Their architecture relies on the combination of high-level protocol specifications (to express how the infrastructure services should be used) and distributed run-time monitoring to regulate the behaviour of third-party applications in the system.

A formal theory for static/dynamic verification Our framework is based on the idea that, if each endpoint is *independently* verified (statically or dynamically) to conform to their local protocols, then the global protocol is respected *as a whole*. To this goal, we propose a new formal model and bisimulation theories of heterogeneous networks of monitored and unmonitored processes.

For the first time, we make explicit the *routing mechanism* implicitly present inside the MPST framework: in a session, messages are sent to abstract roles (e.g. to a Seller) and the router, a dynamically updated component of the network, translates these roles into actual addresses.

By taking this feature into account when designing novel equivalences, our formal model can relate networks built in different ways (through different distributions or relocations of services) but offering the same *interface* to an external observer. The router, being in charge of associating roles with principals, hides to an external user the internal composition of a network: what distinguishes two networks is not their structure but the services they are able to perform, or more precisely, the local types they offer to the outside.

We formally define a satisfaction relation to express when the behaviour of a network conforms to a global specification and we prove a number of properties of our model. *Local safety* states that a monitored process respects its local protocol, i.e. that dynamic verification by monitoring is sound, while *local transparency* states that a monitored process has equivalent behaviour to an unmonitored but well-behaved process, e.g. statically verified against the same local protocol. *Global safety* states that a system satisfies the global protocol, provided that each participant behaves as if monitored, while *global transparency* states that a fully monitored network has equivalent behaviour to an unmonitored but well-behaved network, i.e. in which all local processes are well-behaved against the same local protocols. *Session fidelity* states that, as all message flows of a network satisfy global specifications, whenever the network changes because some local processes take actions, all message flows continue to satisfy global specifications. Together, these properties justify our framework for decentralised verification by allowing monitored and unmonitored processes to be safely mixed while preserving protocol conformance for the entire network. Technically, these properties also ensure the coherence of our theory, by relating the satisfaction relations with the semantics and static validation procedures.

Paper summary and contributions § 2 introduces the formalisms for protocol specifications (§ 2.1) and networks (§ 2.2) used to provide a formal framework for monitored networks based on π-calculus processes and protocol-based runtime enforcement through monitors. § 3 introduces: a semantics for specifications (§ 3.1), a novel behavioural theory for compositional reasoning over monitored networks through the use of equivalences (bisimilarity and barbed congruence) and the satisfaction relation (§ 3.2). § 3.4 establishes key properties of monitored networks, namely local/global safety, transparency, and session fidelity. We discuss future and related work in § 4. The proofs can be found in [2].

2 Types, Processes and Networks: a Formal Presentation

This section and the next one provide a theoretical basis for protocol-centred safety assurance. We first summarise the syntax of MPSTs (multiparty session types) annotated with logical assertions [3]. We then introduce a novel *monitored session calculus* as a variant of the π-calculus, modelling distributed dynamic components (whose behaviours are realised by processes) and monitors, all residing in global networks.

2.1 Multiparty Session Types with Assertions

Multiparty session types with assertions [3] are abstract descriptions of the structure of interactions among the participants of a multiparty session, specifying potential flows of messages, the conditions under which these interactions may be done, and the constraints on the communicated values. In this framework, *global types with assertions*, or just *global types*, describe multiparty sessions

from a network perspective. From global types one can derive, through *endpoint projection*, *local types with assertions*, or just *local types*, describing the protocol from the perspective of a single endpoint.

$$
\begin{array}{l}
A ::= \mathsf{tt} \mid \mathsf{ff} \mid e_1 = e_2 \mid e_1 < e_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \\
e \ ::= v \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 \ \mathtt{mod}\ e_2 \qquad S ::= \mathsf{bool} \mid \mathsf{int} \mid \mathsf{string} \\
G ::= \mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I} \mid G_1 \mid G_2 \mid G_1 ; G_2 \mid \mu \mathsf{t}.G \mid \mathsf{t} \mid \epsilon \mid \mathsf{end} \\
T ::= \mathtt{r}!\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid \mathtt{r}?\{l_i(x_i : S_i)\{A_i\}.T_i\}_{i \in I} \mid T_1 \mid T_2 \mid T_1 ; T_2 \mid \\
\qquad \mu \mathsf{t}.T \mid \mathsf{t} \mid \epsilon \mid \mathsf{end}
\end{array}
$$

The syntax of the global types ($G, G', \ldots$) and local types ($T, T', \ldots$) is given above. The grammar is based on [3, 12] extended with parallel threads, which also require sequential composition to merge parallel threads as in [19]. We let values $v, v', \ldots$ range over boolean constants, numerals and strings, and $e, e', \ldots$ range over first-order expressions. For expressing constraints, we use logical predicates, or *assertions*, ranged over by $A, A', \ldots$, following the grammar given above, although other decidable logics could be used.[1] The *sorts* of exchanged values ($S, S', \ldots$) consists of atomic types.

Global types with assertions $\mathtt{r}_1 \to \mathtt{r}_2 : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i \in I}$ models an interaction where role $\mathtt{r}_1$ sends role $\mathtt{r}_2$ one of the *branch labels* l_i, as well as a value denoted by an *interaction variable* x_i of sort S_i. Interaction variable x_i binds its occurrences in A_i and G_i. A_i is the assertion which needs to hold for $\mathtt{r}_1$ to select l_i, and which may constrain the values instantiating x_i. $G_1 \mid G_2$ specifies two parallel sessions, and $G_1 ; G_2$ denotes sequential composition (assuming that G_1 does not include end). $\mu \mathsf{t}.G$ is a recursive type, where t is guarded in G in the standard way, ϵ is the inaction for absence of communication, and end ends the session.

*Example 1 (*ATM*: the global type).* We present global type $G_{\mathtt{ATM}}$ that specifies an *ATM* scenario. Each session of *ATM* involves three roles: a client ($\mathtt{C}$), the payment server ($\mathtt{S}$) and a separate authenticator ($\mathtt{A}$).

$$
\begin{array}{rl}
G_{\mathtt{ATM}} = & \mathtt{C} \to \mathtt{A} : \{\ \mathtt{Login}(x_i : \mathsf{string})\{\mathsf{tt}\}. \\
& \quad \mathtt{A} \to \mathtt{S} : \{\ \mathtt{LoginOK}()\{\mathsf{tt}\}.\ \mathtt{A} \to \mathtt{C} : \{\mathtt{LoginOK}()\{\mathsf{tt}\}.\ G_{\mathtt{Loop}}\}, \\
& \qquad\quad \mathtt{LoginFail}()\{\mathsf{tt}\}.\ \mathtt{A} \to \mathtt{C} : \{\mathtt{LoginFail}()\{\mathsf{tt}\}.\ \mathsf{end}\}\}\} \\
G_{\mathtt{Loop}} = & \mu\ \mathtt{LOOP}. \\
& \quad \mathtt{S} \to \mathtt{C} : \{\ \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}. \\
& \quad \mathtt{C} \to \mathtt{S} : \{\ \mathtt{Withdraw}(x_p : \mathsf{int})\{x_p > 0 \wedge x_b - x_p \geq 0\}.\ \mathtt{LOOP}, \\
& \qquad\quad \mathtt{Deposit}(x_d : \mathsf{int})\{x_d > 0\}.\ \mathtt{LOOP}, \\
& \qquad\quad \mathtt{Quit}()\{\mathsf{tt}\}.\mathsf{end}\}\}
\end{array}
$$

At the start of the session $\mathtt{C}$ sends its login details x_i to $\mathtt{A}$, then $\mathtt{A}$ informs $\mathtt{S}$ and $\mathtt{C}$ whether the authentication is successful, by choosing either the branch

[1] We use a logic without quantifiers, contrary to [3], to simplify the presentation and because *monitorability*, defined later in this section, makes them unnecessary.

with label LoginOK or LoginFail. In the former case C and S enter a transaction loop specified by $G_{\texttt{Loop}}$. In each iteration S sends C the amount x_b available in the account, which must be non negative. Next, C has three choices: Withdraw withdraws an amount x_p (x_p must be positive and not exceed the current amount x_b) and repeats the loop, Deposit deposits a positive amount x_d in the account and repeats the loop, and Quit ends the session.

We consider global types that satisfy the consistency conditions defined in [11, 3, 12] which rule out, for instance, protocols where interactions have causal relations that cannot be enforced (e.g., we write $\mathtt{r}_A \to \mathtt{r}_B : \mathtt{l}_1()\{\mathsf{tt}\} \mid \mathtt{r}_C \to \mathtt{r}_D : \mathtt{l}_2()\{\mathsf{tt}\}$ instead of $\mathtt{r}_A \to \mathtt{r}_B : \mathtt{l}_1()\{\mathsf{tt}\}.\mathtt{r}_C \to \mathtt{r}_D : \mathtt{l}_2()\{\mathsf{tt}\}$). In addition we assume *monitorability* requiring that in all the interactions of the form $\mathtt{r} \to \mathtt{r}' : \mathtt{l}(x : \mathsf{S})\{A\}$ occurring in a global type G both r and r′ *know* (i.e., have sent or received in a previous or in this interaction) the free variables in A.

Local types with assertions Each local type T is associated with a role taking part in a session. Local type $\mathtt{r}!\{l_i(x_i{:}S_i)\{A_i\}.T_i\}_{i\in I}$ models an interaction where the role under consideration sends r a branch label l_i and a message denoted by an interaction variable x_i of sort S_i. Its dual is the receive interaction $\mathtt{r}?\{l_i(x_i{:}S_i)\{A_i\}.T_i\}_{i\in I}$. The other local types are similar to the global types.

One can derive a set of local types T_i from a global type G by *endpoint projection*, defined as in [3]. We write $G \upharpoonright \mathtt{r}$ for the projection of G onto role r. We illustrate the main projection rule, which is for projecting a global type modelling an interaction. Let G be $(\mathtt{r} \to \mathtt{r}' : \{l_i(x_i : S_i)\{A_i\}.G_i\}_{i\in I})$; the projection of G on r is $\mathtt{r}'!\{l_i(x_i : S_i)\{A_i\}.(G_i \upharpoonright \mathtt{r})\}_{i\in I}$, and the projection of G on r′ is $\mathtt{r}?\{l_i(x_i : S_i)\{A_i\}.(G_i \upharpoonright \mathtt{r}')\}_{i\in I}$. The other rules are homomorphic, following the grammar of global types inductively.

*Example 2 (*ATM*: the local type of C).* We present the *local type* $T_{\mathtt{C}}$ obtained by projecting G_{ATM} on role C.

$$
\begin{array}{l|l}
T_{\mathtt{C}} = \mathtt{A}!\{\mathtt{Login}(x_i : \mathsf{string})\{\mathsf{tt}\}. & T_{\texttt{Loop}} = \mu\, \mathtt{LOOP}. \\
\quad \mathtt{A}?\{\mathtt{LoginOK}()\{\mathsf{tt}\}.\ T_{\texttt{Loop}} & \quad \mathtt{S}?\{\mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}. \\
\quad\quad \mathtt{LoginFail}()\{\mathsf{tt}\}.\ \mathsf{end}\}\} & \quad\quad \mathtt{S}!\{\mathtt{Withdraw}(x_p : \mathsf{int})\{x_p > 0 \wedge x_b - x_p \geq 0\}. \\
 & \quad\quad\quad \mathtt{LOOP}, \\
 & \quad\quad\quad \mathtt{Deposit}(x_d : \mathsf{int})\{x_d > 0\}.\mathtt{LOOP}, \\
 & \quad\quad\quad \mathtt{Quit}()\{\mathsf{tt}\}.\mathsf{end}\}
\end{array}
$$

$T_{\mathtt{C}}$ specifies the behaviour that C should follow to meet the contract of global type G_{ATM}. $T_{\mathtt{C}}$ states that C should first authenticate with A, then receive the Account message from S, and then has the choice of sending Withdraw (and enact the recursion), or Deposit (and enact the recursion) or Quit (and end the session).

2.2 Formal Framework of Processes and Networks

In our formal framework, each distributed application consists of one or more sessions among *principals*. A principal with behaviour P and name α is represented as $[P]_\alpha$. A *network* is a set of principals together with a (unique) *global*

transport, which abstractly represents the communication functionality of a distributed system. The syntax of processes, principals and networks is given below, building on the multiparty session π-calculus from [1].

$$P ::= \overline{a}\langle s[\mathtt{r}] : T\rangle \mid a(y[\mathtt{r}]{:}T).P \mid k[\mathtt{r}_1, \mathtt{r}_2]!l\langle e\rangle \mid k[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_{i\in I} \mid$$
$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid P \mid Q \mid \mathbf{0} \mid \mu X.P \mid X \mid P;Q \mid (\nu a)\ P \mid (\nu s)P$$
$$N ::= [P]_\alpha \mid N_1 \mid N_2 \mid \mathbf{0} \mid (\nu a)N \mid (\nu s)N \mid \langle r\ ;\ h\rangle$$
$$r ::= a \mapsto \alpha \mid s[\mathtt{r}] \mapsto \alpha \qquad h ::= m \cdot h \mid \emptyset \qquad m ::= \overline{a}\langle s[\mathtt{r}] : T\rangle \mid s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle$$

$\mathtt{r}, \mathtt{r}_1, \ldots$ roles	$s, s', \ldots$ session names	$X, Y, \ldots$ process variables
$a, b, \ldots$ shared names	$x, y, \ldots$ variables	$P, Q, \ldots$ processes
$\alpha, \beta, \ldots$ principal names	$N, N', \ldots$ networks	

Processes Processes are ranged over by $P, P', \ldots$ and communicate using two types of channel: *shared channels* (or shared names) used by processes for sending and receiving invitations to participate in sessions, and *session channels* (or session names) used for communication *within* established sessions. One may consider session names as e.g., URLs or service names.

The *session invitation* $\overline{a}\langle s[\mathtt{r}] : T\rangle$ invites, through a shared name a, another process to play $\mathtt{r}$ in a session s. The *session accept* $a(y[\mathtt{r}] : T).P$ receives a session invitation and, after instantiating y with the received session name, behaves in its continuation P as specified by local type T for role $\mathtt{r}$. The *selection* $k[\mathtt{r}_1, \mathtt{r}_2]!l\langle e\rangle$ sends, through session channel k (of an established session), and as a sender $\mathtt{r}_1$ and to a receiver $\mathtt{r}_2$, an expression e with label l. The *branching* $k[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_{i\in I}$ is ready to receive one of the labels and a value, then behaves as P_i after instantiating x_i with the received value. We omit labels when I is a singleton. The *conditional*, *parallel* and *inaction* are standard. The *recursion* $\mu X.P$ defines X as P. Processes $(\nu a)P$ and $(\nu s)P$ hide shared names and session names, respectively.

Principals and network A *principal* $[P]_\alpha$, with its process P and name α, represents a unit of behaviour (hence verification) in a distributed system. A *network* N is a collection of principals with a unique global transport.

A *global transport* $\langle r\ ;\ h\rangle$ is a pair of a routing table which delivers messages to principals, and a global queue. Messages between two parties inside a single session are ordered (as in a TCP connection), otherwise unordered. More precisely, in $\langle r\ ;\ h\rangle$, h is a *global queue*, which is a sequence of *messages* $\overline{a}\langle s[\mathtt{r}] : T\rangle$ or $s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle$, ranged over by m. These m represent messages-in-transit, i.e. those messages which have been sent from some principals but have not yet been delivered. The *routing table* r is a finite map from session-roles and shared names to principals. If, for instance, $s[\mathtt{r}] \mapsto \alpha \in r$ then a message for $\mathtt{r}$ in session s will be delivered to principal α.

Let $n, n', \ldots$ range over shared and session channels. A network N which satisfies the following conditions is *well-formed*: (1) N contains at most one

global transport; (2) two principals in N never have the same principal name; and (3) if $N \equiv (\nu\tilde{n})(\prod_i [P_i]_{\alpha_i} | \langle r \ ; \ h \rangle)$, each free shared or session name in P_i and h occurs in $\tilde{n}$ (we use $\prod_i P_i$ to denote $P_1 \mid P_2 \cdots \mid P_n$).

Semantics The reduction relation for dynamic networks is generated from the rules below, which model the interactions of principals with the global queue.

$$[\overline{a}\langle s[\mathtt{r}] : T\rangle]_\alpha \mid \langle r \ ; \ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \langle r \ ; \ h \cdot \overline{a}\langle s[\mathtt{r}] : T\rangle\rangle \qquad \lfloor \textsc{req} \rfloor$$

$$[a(y[\mathtt{r}] : T).P]_\alpha \mid \langle r \ ; \ \overline{a}\langle s[\mathtt{r}] : T\rangle \cdot h\rangle \longrightarrow [P[s/y]]_\alpha \mid \langle r \cdot s[\mathtt{r}] \mapsto \alpha \ ; \ h\rangle^{\ \dagger} \qquad \lfloor \textsc{acc} \rfloor$$

$$[s[\mathtt{r}_1, \mathtt{r}_2]! l_j\langle v\rangle]_\alpha \mid \langle r \ ; \ h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \langle r \ ; \ h \cdot s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle\rangle^{\ \dagger\dagger} \qquad \lfloor \textsc{sel} \rfloor$$

$$[s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \langle r \ ; \ s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle \cdot h\rangle \longrightarrow [P_j[v/x_j]]_\alpha \mid \langle r \ ; \ h\rangle^{\ \dagger\dagger\dagger} \qquad \lfloor \textsc{bra} \rfloor$$

$$[\mathsf{if}\ \mathsf{tt}\ \mathsf{then}\ P\ \mathsf{else}\ Q]_\alpha \longrightarrow [P]_\alpha \qquad [\mathsf{if}\ \mathsf{ff}\ \mathsf{then}\ P\ \mathsf{else}\ Q]_\alpha \longrightarrow [Q]_\alpha \qquad \lfloor \textsc{cnd} \rfloor$$

$$\frac{[P]_\alpha \mid N \longrightarrow [P']_\alpha \mid N'}{[\mathcal{E}(P)]_\alpha \mid N \longrightarrow [\mathcal{E}(P')]_\alpha \mid N'} \qquad \frac{e \longrightarrow e'}{[\mathcal{E}(e)]_\alpha \longrightarrow [\mathcal{E}(e')]_\alpha} \qquad \frac{N \longrightarrow N'}{\mathcal{E}(N) \longrightarrow \mathcal{E}(N')} \qquad \lfloor \textsc{ctx} \rfloor$$

$$\dagger : r(a) = \alpha \qquad \dagger\dagger : r(s[\mathtt{r}_2]) \neq \alpha \qquad \dagger\,\dagger\dagger : r(s[\mathtt{r}_2]) = \alpha$$

$$\mathcal{E} ::= (\ ) \ \mid \ \mathcal{E} \mid P \ \mid \ (\nu s)\mathcal{E} \mid (\nu a)\mathcal{E} \mid \mathcal{E}; P \mid \mathcal{E} \mid N \mid \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ P\ \mathsf{else}\ Q \mid s[\mathtt{r}_1, \mathtt{r}_2]! l\langle \mathcal{E}\rangle$$

Rule $\lfloor \textsc{req} \rfloor$ places an invitation in the global queue. Dually, in $\lfloor \textsc{acc} \rfloor$, a process receives an invitation on a shared name from the global queue, assuming a message on a is to be routed to α. As a result, the routing table adds $s[\mathtt{r}] \mapsto \alpha$ in the entry for s. Rule $\lfloor \textsc{sel} \rfloor$ puts in the queue a message sent from $\mathtt{r}_1$ to $\mathtt{r}_2$, which selects label l_j and carries v, if it is not going to be routed to α (i.e. sent to self). Dually, $\lfloor \textsc{bra} \rfloor$ gets a message with label l_j from the global queue, so that the j-th process P_j receives value v. The reduction is also defined modulo the structural congruence $\equiv$ defined by the standard laws over processes/networks, the unfolding of recursion ($\mu X.P \equiv P[\mu X.P/X]$) and the associativity and commutativity and the rules of message permutation in the queue [15, 11]. The other rules are standard.

*Example 3 (*ATM*: an implementation).* We now illustrate the processes implementing the client role of the *ATM* protocol. We let $P_{\mathtt{C}}$ be the process implementing $T_{\mathtt{C}}$ (from Example 2) and communicating on session channel s.

$$\begin{array}{l|l} P_{\mathtt{C}} = s[\mathtt{C}, \mathtt{A}]!\mathtt{Login}(alice\_pwd123); & P''_{\mathtt{C}} = \mathsf{if}\ getmore() \wedge (x_b \geq 10) \\ \quad s[\mathtt{A}, \mathtt{C}]?\{\mathtt{LoginOK}(); \mu X.P'_{\mathtt{C}}, \mathtt{LoginFail}().\mathbf{0}\} & \quad \mathsf{then}\ s[\mathtt{C}, \mathtt{S}]!\mathtt{Withdraw}(10); X \\ P'_{\mathtt{C}} = s[\mathtt{S}, \mathtt{C}]?\mathtt{Account}(x_b); P''_{\mathtt{C}} & \quad \mathsf{else}\ s[\mathtt{C}, \mathtt{S}]!\mathtt{Quit}(); \mathbf{0} \end{array}$$

Note that $P_{\mathtt{C}}$ selects only two of the possible branches (i.e., `Withdraw` and `Quit`) and `Deposit` is never selected. One can think of $P_{\mathtt{C}}$ as an ATM machine that only allows to withdraw a number of £10 banknotes, until the amount exceeds the current balance. This ATM machine does not allow deposits. We assume *getmore*() to be a local function to the principal running $P_{\mathtt{C}}$ that returns tt if more notes are required (ff otherwise). $P_{\mathtt{S}}$ below implements the server role:

$$\begin{array}{l|l} P_{\mathtt{S}} = s[\mathtt{A}, \mathtt{S}]?\{\mathtt{LoginOK}(); \mu X.P'_{\mathtt{S}}, \mathtt{LoginFail}().\mathbf{0}\} & P''_{\mathtt{S}} = s[\mathtt{C}, \mathtt{S}]?\{\mathtt{Withdraw}(x_p).X, \\ P'_{\mathtt{S}} = s[\mathtt{S}, \mathtt{C}]!\mathtt{Account}(getBalance()); P''_{\mathtt{S}} & \qquad\qquad \mathtt{Deposit}(x_d).X, \\ & \qquad\qquad \mathtt{Quit}().\mathbf{0}\ \} \end{array}$$

where *getBalance*() is a local function to the principal running $P_{\mathtt{S}}$ that synchronously returns the current balance of the client.

3 Theory of Dynamic Safety Assurance

In this section we formalise the specifications (based on local types) used to guard the runtime behaviour of the principals in a network. These specifications can be embedded into system monitors, each wrapping a principal to ensure that the ongoing communication conforms to the given specification. Then, we present a behavioural theory for monitored networks and its safety properties.

3.1 Semantics of Global Specifications

The specification of the (correct) behaviour of a principal consists of an *assertion environment* $\langle \Gamma; \Delta \rangle$, where Γ is the *shared environment* describing the behaviour on shared channels, and Δ is the *session environment* representing the behaviour on session channels (i.e., describing the sessions that the principal is currently participating in). The syntax of Γ and Δ is given by:

$$\Gamma ::= \emptyset \;\mid\; \Gamma, a : \mathtt{I}(T[\mathtt{r}]) \mid \Gamma, a : \mathtt{O}(T[\mathtt{r}]) \qquad \Delta ::= \emptyset \;\mid\; \Delta, s[\mathtt{r}] : T$$

In Γ, the assignment $a : \mathtt{I}(T[\mathtt{r}])$ (resp. $a : \mathtt{O}(T[\mathtt{r}])$) states that the principal can, through a, receive (resp. send) invitations to play role $\mathtt{r}$ in a session instance specified by T. In Δ, we write $s[\mathtt{r}] : T$ when the principal is playing role $\mathtt{r}$ of session s specified by T. Networks are monitored with respect to *collections of specifications* (or just *specifications*) one for each principal in the network. A specification $\Sigma, \Sigma', \ldots$ is a finite map from principals to assertion environments:

$$\Sigma \;::=\; \emptyset \;\mid\; \Sigma, \alpha : \langle \Gamma; \Delta \rangle$$

The semantics of Σ is defined using the following labels:

$$\ell ::= \overline{a}\langle s[\mathtt{r}] : T \rangle \;\mid\; a\langle s[\mathtt{r}] : T \rangle \;\mid\; s[\mathtt{r}_1, \mathtt{r}_2]!l\langle v \rangle \;\mid\; s[\mathtt{r}_1, \mathtt{r}_2]?l\langle v \rangle \;\mid\; \tau$$

The first two labels are for *invitation* actions, the first is for requesting and the second is for accepting. Labels with $s[\mathtt{r}_1, \mathtt{r}_2]$ indicate *interaction* actions for sending (!) or receiving (?) messages within sessions. The labelled transition relation for specification is defined by the rules below.

$$\alpha : \langle \Gamma, a : \mathtt{O}(T[\mathtt{r}]); \Delta \rangle \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T \rangle} \alpha : \langle \Gamma, a : \mathtt{O}(T[\mathtt{r}]); \Delta \rangle \qquad \text{[Req]}$$

$$\frac{s \notin \mathsf{dom}(\Delta)}{\alpha : \langle \Gamma, a : \mathtt{I}(T[\mathtt{r}]); \Delta \rangle \xrightarrow{a\langle s[\mathtt{r}]:T \rangle} \alpha : \langle \Gamma, a : \mathtt{I}(T[\mathtt{r}]); \Delta, s[\mathtt{r}] : T \rangle} \qquad \text{[Acc]}$$

$$\frac{\Gamma \vdash v : S_j, \quad A_j[v/x_j] \downarrow \mathsf{tt}, \quad j \in I}{\alpha : \langle \Gamma; \Delta, s[\mathtt{r}_2] : \mathtt{r}_1?\{l_i(x_i : S_i)\{A_i\}.T'_i\}_{i \in I} \rangle \xrightarrow{s[\mathtt{r}_1, \mathtt{r}_2]?l_j\langle v \rangle} \alpha : \langle \Gamma; \Delta, s[\mathtt{r}_2] : T'_j[v/x_j] \rangle} \qquad \text{[Bra]}$$

$$\frac{\Gamma \vdash v : S_j, \quad A_j[v/x_j] \downarrow \mathsf{tt}, \quad j \in I}{\alpha : \langle \Gamma; \Delta, s[\mathtt{r}_1] : \mathtt{r}_2!\{l_i(x_i : S_i)\{A_i\}.T'_i\}_{i \in I} \rangle \xrightarrow{s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v \rangle} \alpha : \langle \Gamma; \Delta, s[\mathtt{r}_1] : T'_j[v/x_j] \rangle} \qquad \text{[Sel]}$$

$$\frac{\alpha : \langle \Gamma_1; \Delta_1 \rangle \xrightarrow{\ell} \alpha : \langle \Gamma'_1; \Delta'_1 \rangle}{\alpha : \langle \Gamma_1; \Delta_1 | \Delta_2 \rangle \xrightarrow{\ell} \alpha : \langle \Gamma'_1; \Delta'_1 | \Delta_2 \rangle} \qquad \Sigma \xrightarrow{\tau} \Sigma \qquad \frac{\Sigma_1 \xrightarrow{\ell} \Sigma_2}{\Sigma_1, \Sigma_3 \xrightarrow{\ell} \Sigma_2, \Sigma_3} \qquad \text{[Spl,Tau,Par]}$$

Rule [REQ] allows α to send an invitation on a properly typed shared channel a (i.e., given that the shared environment maps a to $T[\mathtt{r}]$). Rule [ACC] allows α to receive an invitation to be role $\mathtt{r}$ in a new session s, on a properly typed shared channel a. Rule [BRA] allows α, participating to sessions s as $\mathtt{r}_2$, to receive a message with label l_j from $\mathtt{r}_1$, given that A_j is satisfied after replacing x_j with the received value v. After the application of this rule the specification is T_j. Rule [SEL] is the symmetric (output) counterpart of [BRA]. We use $\downarrow$ to denote the evaluation of a logical assertion. [SPL] is the parallel composition of two session environments where $\Delta_1|\Delta_2$ composes two local types: $\Delta_1|\Delta_2 = \{s[\mathtt{r}] : (T_1 \mid T_2) \mid T_i = \Delta_i(s[\mathtt{r}]), s[\mathtt{r}] \in \mathsf{dom}(\Delta_1) \cap \mathsf{dom}(\Delta_2)\} \cup \mathsf{dom}(\Delta_1)/\mathsf{dom}(\Delta_2) \cup \mathsf{dom}(\Delta_2)/\mathsf{dom}(\Delta_1)$. [TAU] says that the specification should be invariant under reduction of principals. [PAR] says if Σ_1 and Σ_3 are composable, after Σ_1 becomes as Σ_2, they are still composable.

3.2 Semantics of Dynamic Monitoring

The endpoint monitor $\mathsf{M}, \mathsf{M}', ...$ for principal α is a specification $\alpha : \langle \Gamma ; \Delta \rangle$ used to *dynamically* ensure that the messages to and from α are legal with respect to Γ and Δ. A *monitored network* N is a network N with monitors, obtained by extending the syntax of networks as:

$$\mathsf{N} ::= N \;\big|\; \mathsf{M} \;\big|\; \mathsf{N} \mid \mathsf{N} \;\big|\; (\nu s)\mathsf{N} \;\big|\; (\nu a)\mathsf{N}$$

The reduction rules for monitored networks are given below and use, in the premises, the labelled transitions of monitors. The labelled transitions of a monitor are the labelled transitions of its corresponding specification (given in § 3.1).

$$\lceil\text{REQ}\rceil \frac{\mathsf{M} \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle} \mathsf{M}'}{[\overline{a}\langle s[\mathtt{r}] : T\rangle]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M}' \mid \langle r \;;\; h \cdot \overline{a}\langle s[\mathtt{r}] : T\rangle\rangle}$$

$$\lceil\text{ACC}\rceil \frac{\mathsf{M} \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} \mathsf{M}' \qquad r(a) = \alpha}{[a(y[\mathtt{r}] : T).P]_\alpha \mid \mathsf{M} \mid \langle r \;;\; \overline{a}\langle s[\mathtt{r}] : T\rangle \cdot h\rangle \longrightarrow [P[s/y]]_\alpha \mid \mathsf{M}' \mid \langle r \cdot s[\mathtt{r}] \mapsto \alpha \;;\; h\rangle}$$

$$\lceil\text{BRA}\rceil \frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l_j\langle v\rangle} \mathsf{M}' \qquad r(s[\mathtt{r}_2]) = \alpha}{[s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i).P_i\}_i]_\alpha \mid \mathsf{M} \mid \langle r \;;\; s\langle \mathtt{r}_1, \mathtt{r}_2, l_j\langle v\rangle\rangle \cdot h\rangle \longrightarrow [P_j[v/x_j]]_\alpha \mid \mathsf{M}' \mid \langle r \;;\; h\rangle}$$

$$\lceil\text{SEL}\rceil \frac{\mathsf{M} \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle} \mathsf{M}' \quad r(s[\mathtt{r}_2]) \neq \alpha}{[s[\mathtt{r}_1, \mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M}' \mid \langle r \;;\; h \cdot s\langle \mathtt{r}_1, \mathtt{r}_2, l\langle v\rangle\rangle\rangle}$$

$$\lceil\text{REQER}\rceil \frac{\mathsf{M} \not\xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle}}{[\overline{a}\langle s[\mathtt{r}] : T\rangle]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle}$$

$$\lceil\text{ACCER}\rceil \frac{\mathsf{M} \not\xrightarrow{a\langle s[\mathtt{r}]:T\rangle}}{[a(y[\mathtt{r}] : T).P]_\alpha \mid \mathsf{M} \mid \langle r \;;\; \overline{a}\langle s[\mathtt{r}] : T\rangle \cdot h\rangle \longrightarrow [a(y[\mathtt{r}] : T).P]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle}$$

$$\lceil\text{SELER}\rceil \frac{\mathsf{M} \not\xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l\langle v\rangle}}{[s[\mathtt{r}_1, \mathtt{r}_2]!l\langle v\rangle]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle \longrightarrow [\mathbf{0}]_\alpha \mid \mathsf{M} \mid \langle r \;;\; h\rangle}$$

The first four rules model reductions that are allowed by the monitor (i.e., in the premise). Rule ⌈REQ⌉ inserts an invitation in the global queue. Rule ⌈ACC⌉ is symmetric and updates the router so that all messages for role $\mathtt{r}$ in session

s will be routed to α. Similarly, ⌈Bra⌉ (resp. ⌈Sel⌉) extracts (resp. introduces) messages from (resp. in) the global queue. The error cases for ⌈Req⌉ and ⌈Sel⌉, namely ⌈ReqEr⌉ and ⌈SelEr⌉, 'skip' the current action (removing it from the process), do not modify the queue, the router nor the state of the monitor. The error cases for ⌈Acc⌉ and ⌈Bra⌉, namely ⌈AccEr⌉ and ⌈BraEr⌉ (the latter omitted for space constraint), do not affect the process, which remains ready to perform the action, and remove the violating message from the queue.

Example 4 (ATM: a monitored network). We illustrate the monitored networks for the ATM scenario, where the routing table is defined as

$$r = a \mapsto \alpha, b \mapsto \beta, c \mapsto \gamma, s[\mathtt{S}] \mapsto \alpha, s[\mathtt{C}] \mapsto \beta, s[\mathtt{A}] \mapsto \gamma$$

We consider the fragment of session where the authentication has occurred, the process of $\mathtt{C}$ (resp. $\mathtt{S}$) is $P'_{\mathtt{C}}$ (resp. $P'_{\mathtt{S}}$) from Example 3, and the process of $\mathtt{A}$ is $\mathbf{0}$.

$\mathsf{N}_{\mathtt{S}} = [P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}} = [s[\mathtt{S},\mathtt{C}]!\ \mathtt{Account}(100); P''_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}}$ (assuming $getBalance() = 100$)
$\mathsf{N}_{\mathtt{C}} = [P'_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}} = [s[\mathtt{S},\mathtt{C}]?\ \mathtt{Account}(x_b).P''_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}}$
$\mathsf{N}_{\mathtt{A}} = [\mathbf{0}]_\gamma \mid \gamma : \langle c : T_{\mathtt{A}}[\mathtt{A}]\ ;\ s[\mathtt{A}] :\ \mathsf{end}\rangle$

where $\mathsf{M}_{\mathtt{S}} = \alpha : \langle a :\ T_{\mathtt{S}}[\mathtt{S}]\ ;\ s[\mathtt{S}] :\ \mathtt{C}!\ \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}.T'_{\mathtt{S}}\rangle$ and $\mathsf{M}_{\mathtt{C}}$ is dual.

$$\mathsf{N}_1 = [s[\mathtt{S},\mathtt{C}]!\ \mathtt{Account}(100); P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}_{\mathtt{S}} \mid [s[\mathtt{S},\mathtt{C}]?\ \mathtt{Account}(x_b).P'_{\mathtt{C}}]_\beta \mid \mathsf{M}_{\mathtt{C}} \mid \mathsf{N}_{\mathtt{A}} \mid \langle r\ ;\ \emptyset\rangle$$
$$\longrightarrow\longrightarrow [P'_{\mathtt{S}}]_\alpha \mid \mathsf{M}'_{\mathtt{S}} \mid [P'_{\mathtt{C}}[100/x_b]]_\beta \mid \mathsf{M}'_{\mathtt{C}} \mid \mathsf{N}_{\mathtt{A}} \mid \langle r\ ;\ \emptyset\rangle$$

where $\mathsf{M}'_{\mathtt{S}} = \alpha : \langle a : T_{\mathtt{S}}[\mathtt{S}]\ ;\ s[\mathtt{S}] :\ T'_{\mathtt{S}}\rangle$ and $\mathsf{M}'_{\mathtt{C}} = \beta : \langle b : T_{\mathtt{C}}[\mathtt{C}]\ ;\ s[\mathtt{C}] :\ T'_{\mathtt{C}}\rangle$

Above, $x_b \geq 0$ is satisfied since $x_b = 100$. If the server tried to communicate e.g., value -100 for x_b, the monitoring (by rule ⌈SelEr⌉) would drop the message.

3.3 Network Satisfaction and Equivalences

Based on the formal representations of monitored networks, we now introduce the key formal tools for analysing their behaviour. First, we introduce *bisimulation* and *barbed congruence* over networks, and develop the notion of *interface*. Then we define *the satisfaction relation* $\models N : \mathsf{M}$, used in § 3.4 to prove the properties of our framework.

Bisimulations We use $M, M', ...$ for a *partial network*, that is a network which does not contain a global transport, hence enabling the global observation of interactions. The labelled transition relation for processes and partial networks M is defined below.

(Req) $[\overline{a}\langle s[\mathtt{r}] : T\rangle; P]_\alpha \xrightarrow{\overline{a}\langle s[\mathtt{r}]:T\rangle} [\mathbf{0}]_\alpha$ (Acc) $[a(y[\mathtt{r}] : T).P]_\alpha \xrightarrow{a\langle s[\mathtt{r}]:T\rangle} [P[s/y]]_\alpha$

(Bra) $[s[\mathtt{r}_1, \mathtt{r}_2]?\{l_i(x_i : S_i).P_i\}_i]_\alpha \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]?l_j\langle v\rangle} [P_j[v/x_j]]_\alpha$

(Sel) $[s[\mathtt{r}_1, \mathtt{r}_2]!l_j\langle v\rangle]_\alpha \xrightarrow{s[\mathtt{r}_1,\mathtt{r}_2]!l_j\langle v\rangle} [\mathbf{0}]_\alpha$ (ctx) $\dfrac{[P]_\alpha \xrightarrow{\ell} [P']_\alpha \ \ \mathsf{n}(\ell) \cap \mathsf{bn}(\mathcal{E}) = \emptyset}{[\mathcal{E}(P)]_\alpha \xrightarrow{\ell} [\mathcal{E}(P')]_\alpha}$

(tau) $\dfrac{M \longrightarrow M'}{M \xrightarrow{\tau} M'}$ (res) $\dfrac{M \xrightarrow{\ell} M' \ a \notin \mathsf{sbj}(\ell)}{(\nu a)M \xrightarrow{\ell\setminus a} (\nu a)M'}$ (str) $\dfrac{M \equiv M_0 \xrightarrow{\ell} M'_0 \equiv M'}{M \xrightarrow{\ell} M'}$

In (CTX), $\mathsf{n}(\ell)$ indicates the names occurring in ℓ while $\mathsf{bn}(\mathcal{E})$ indicates binding $\mathcal{E}$ induces. In (RES), $\mathsf{sbj}(\ell)$ denotes the subject of ℓ. In (TAU) the axiom is obtained either from the reduction rules for dynamic networks given in § 2.2 (only those not involving the global transport), or from the corresponding rules for monitored networks (which have been omitted in § 3.2).

Hereafter we write $\Longrightarrow$ for $\stackrel{\tau}{\rightarrow}^*$, $\stackrel{\ell}{\Longrightarrow}$ for $\Longrightarrow\stackrel{\ell}{\rightarrow}\Longrightarrow$, and $\stackrel{\hat{\ell}}{\Longrightarrow}$ for $\Longrightarrow$ if $\ell = \tau$ and $\stackrel{\ell}{\Longrightarrow}$ otherwise.

Definition 1 (Bisimulation over partial networks). A binary relation $\mathcal{R}$ over partial networks is a *weak bisimulation* when $M_1 \mathcal{R} M_2$ implies: whenever $M_1 \stackrel{\ell}{\rightarrow} M_1'$ such that $\mathsf{bn}(\ell) \cap \mathsf{fn}(M_2) = \emptyset$, we have $M_2 \stackrel{\hat{\ell}}{\Longrightarrow} M_2'$ such that $M_1' \mathcal{R} M_2'$, and the symmetric case. We write $M_1 \approx M_2$ if (M_1, M_2) are in a weak bisimulation.

Interface We want to build a model where two different implementations of the same service are related. Bisimilarity is too strong for this aim (as shown in Example 5). We use instead a contextual congruence (barbed reduction-closed congruence [14]) $\cong$ for networks. Intuitively, two networks are barbed-congruent when they are indistinguishable for any principal that connects to them. In this case we say they propose the same *interface* to the exterior. Formally, two networks are related with $\cong$ when, composed with the same third network, they offer the same *barbs* (the messages to external principals in the respective global queues are on the same channels) and this property is preserved under reduction.

We say that a message m is *routed for* α *in* N if $N = (\nu \tilde{n})(M_0 \mid \langle r \ ; \ h \rangle)$, $m \in h$, either $m = \overline{a}\langle s[\mathtt{r}] : T \rangle$ and $r(a) = \alpha$ or $m = s[\mathtt{r}_1, \mathtt{r}_2]!l\langle e \rangle$ and $r(s[\mathtt{r}_2]) = \alpha$.

Definition 2 (Barb). We write $N \downarrow_a$ when the global queue of N contains a message m to free a and m is routed for a principal not in N. We write $N \Downarrow_a$ if $N \longrightarrow^* N' \downarrow_a$.

We denote $\mathcal{P}(N)$ for a set of principals in N, $\mathcal{P}(\prod [P_i]_{\alpha_i}) = \{\alpha_1, ..., \alpha_n\}$. We say N_1 and N_2 are *composable* when $\mathcal{P}(N_1) \cap \mathcal{P}(N_2) = \emptyset$, the union of their routing tables remains a function, and their free session names are disjoint. If N_1 and N_2 are composable, we define $N_1 \diamond N_2 = (\nu \tilde{n}_1, \tilde{n}_2)(M_1 \mid M_2 \mid \langle r_1 \cup r_2 \ ; \ h_1 \cdot h_2 \rangle)$ where $N_i = (\nu \tilde{n}_i)(M_i \mid \langle r_i \ ; \ h_i \rangle)$ $(i = 1, 2)$. Notice that both equivalences are compositional, as proved in Proposition § 4.

Definition 3 (Barbed reduction-closed congruence). A relation $\mathcal{R}$ on networks with the same principals is a *barbed r.c. congruence* [14] if the following holds: whenever $N_1 \mathcal{R} N_2$ we have: (1) for each composable N, $N \diamond N_1 \mathcal{R} N \diamond N_2$; (2) $N_1 \longrightarrow N_1'$ implies $N_2 \longrightarrow^* N_2'$ s.t. $N_1' \mathcal{R} N_2'$ again, and the symmetric case; (3) $N_1 \Downarrow_a$ iff $N_2 \Downarrow_a$. We write $N_1 \cong N_2$ when they are related by a barbed r.c. congruence.

The following result states that composing two bisimilar partial networks with the same network – implying the same router and global transport – yields two undistinguishable networks.

Proposition 4 (Congruency). If $M_1 \approx M_2$, then (1) $M_1|M \approx M_2|M$ for each composable partial M; and (2) $M_1|N \cong M_2|N$ for each composable N.

Example 5 (ATM: an example of behavioural equivalence). We use an example to illustrate our notion of interface. As our verification by monitors is done separately for each endpoint, one can safely modify a global specification as long as its projection on the public roles stays the same. The barbed congruence we introduce takes this into account: two networks proposing the same service, but organised in different ways, are equated even if the two networks correspond to different global specifications. As an example, consider global type $G^2_{\mathtt{ATM}}$ defined as $G_{\mathtt{ATM}}$ where $G^2_{\mathtt{Loop}}$ is used in place of $G_{\mathtt{Loop}}$ from Example 3. $G^2_{\mathtt{Loop}}$ involves a fourth party, the transaction agent `B`: `S` sends a query to `B` which gives back a one-use transaction identifier. Then, the protocol proceeds as the original one. Notably, $G_{\mathtt{ATM}}$ and $G^2_{\mathtt{ATM}}$ have the same interfaces for the client (resp. the authenticator), as their projections of on `C` (resp. `A`) are equal.

$$
\begin{array}{l}
G^2_{\mathtt{Loop}} = \mu\, \mathtt{LOOP}. \\
\quad \mathtt{S} \to \mathtt{B} : \{\ \mathtt{Query}()\{\mathsf{true}\}. \\
\quad \mathtt{B} \to \mathtt{S} : \{\ \mathtt{Answer}(x_t : \mathsf{int})\{\mathsf{true}\}. \\
\quad \mathtt{S} \to \mathtt{C} : \{\ \mathtt{Account}(x_b : \mathsf{int})\{x_b \geq 0\}. \\
\quad \mathtt{C} \to \mathtt{S} : \{\ \mathtt{Withdraw}(x_p : \mathsf{int})\{x_p \geq 0 \wedge x_b - x_p \geq 0\}.\ \mathtt{LOOP}, \\
\qquad\qquad \mathtt{Deposit}(x_d : \mathsf{int})\{x_d > 0\}.\ \mathtt{LOOP}, \\
\qquad\qquad \mathtt{Quit}()\{\mathsf{true}\}.\mathtt{end} \qquad\qquad \}\}\}\}
\end{array}
$$

We define $P^2_{\mathtt{S}}$ as $P_{\mathtt{S}}$ in Example 3 but replacing the occurrence of $P'_{\mathtt{S}}$ in $P_{\mathtt{S}}$ by

$$s[\mathtt{S},\mathtt{B}]!\mathtt{Query}\langle\rangle; s[\mathtt{B},\mathtt{S}]?\mathtt{Answer}(x_t).P'_{\mathtt{S}}$$

and also $N^2_{\mathtt{S}} = [P^2_{\mathtt{S}}]_\alpha$ and $N_{\mathtt{B}} = [\mu X.s[\mathtt{S},\mathtt{B}]?\mathtt{Query}\langle\rangle; s[\mathtt{B},\mathtt{S}]!\mathtt{Answer}\langle getTrans()\rangle]_\delta$. By definition, the two following networks are barbed-congruent:

$$
\begin{array}{l}
(N_{\mathtt{S}} \mid \langle \emptyset\ ;\ s[\mathtt{S}] \mapsto \alpha, s[\mathtt{C}] \mapsto \beta, s[\mathtt{A}] \mapsto \gamma\rangle) \cong \\
\qquad (N^2_{\mathtt{S}} \mid N_{\mathtt{B}} \mid \langle \emptyset\ ;\ s[\mathtt{S}] \mapsto \alpha, s[\mathtt{C}] \mapsto \beta, s[\mathtt{A}] \mapsto \gamma, s[\mathtt{B}] \mapsto \delta\rangle)
\end{array}
$$

even if the first one implements the original ATM protocol while the second one implements its variant. Indeed, composed with any tester, such as $N_{\mathtt{C}} \mid N_{\mathtt{A}} = [P_{\mathtt{C}}]_\beta \mid [P_{\mathtt{A}}]_\gamma$ these two networks will produce the same interactions.

However, the corresponding partial networks $N^2_{\mathtt{S}} \mid N_{\mathtt{B}}$ and $N_{\mathtt{S}}$ are not bisimilar: the former is able to perform a transition labelled $s[\mathtt{S},\mathtt{B}]!\mathtt{Query}\langle\rangle$ while the latter is not. This difference in behaviour is not visible to the barbed congruence, as it takes into account the router which prevents the messages $s[\mathtt{S},\mathtt{B}]!\mathtt{Query}\langle\rangle$ to be caught by a tester. As an example of network bisimilar to $N_{\mathtt{S}}$, consider:

$$N_1 = (\nu k)\ ([P_{\mathtt{S}} \mid P_{\mathtt{S}}[k/s]]_\alpha \ \mid\ [P_{\mathtt{C}}[k/s]]_\delta)$$

In this partial network, principal α plays both `S` in public session s (as in $N_{\mathtt{S}}$) and `S` in the private session k. Principal δ plays `C` in the latter. As k is private, N_1 offers the same observable behaviour than $N_{\mathtt{S}}$ (no action on k can be observed), and we have $N_1 \approx N_{\mathtt{S}}$.

Satisfaction We present a satisfaction relation for partial networks, which include local principals. If M is a partial network, $\models M : \Sigma$ s.t. $\mathsf{dom}(\Sigma) = \mathcal{P}(M)$, means that the specification allows all outputs from the network; that the network is ready to receive all the inputs indicated by the specification; and that this is preserved by transition.

Definition 5 (Satisfaction). Let $\mathsf{sbj}(\ell)$ denote the subject of $\ell \neq \tau$. A relation $\mathcal{R}$ from partial networks to specifications is a *satisfaction* when $M\mathcal{R}\Sigma$ implies:

1. If $\Sigma \xrightarrow{\ell} \Sigma'$ for an input ℓ and M has an input at $\mathsf{sbj}(\ell)$, then $M \xrightarrow{\ell} M'$ s.t. $M'\mathcal{R}\Sigma'$.
2. If $M \xrightarrow{\ell} M'$ for an output at ℓ, then $\Sigma \xrightarrow{\ell} \Sigma'$ s.t. $M'\mathcal{R}\Sigma'$.
3. If $M \xrightarrow{\tau} M'$, then $\Sigma \xrightarrow{\tau} \Sigma'$ s.t. $M'\mathcal{R}\Sigma'$ (i.e. $M'\mathcal{R}\Sigma$ since $\Sigma \xrightarrow{\tau} \Sigma$ always).

When $M\mathcal{R}\Sigma$ for a satisfaction relation $\mathcal{R}$, we say M *satisfies* Σ, denoted $\models M : \Sigma$. By Definition 5 and Proposition 4 we obtain:

Proposition 6. If $M_1 \cong M_2$ and $\models M_1 : \Sigma$ then $\models M_2 : \Sigma$.

3.4 Safety Assurance and Session Fidelity

In this section, we present the properties underpinning safety assurance in the proposed framework from different perspectives.

Theorem 7 shows local safety/transparency, and global safety/transparency for fully monitored networks. A network N is *fully monitored wrt* Σ when all its principals are monitored and the collection of the monitors is congruent to Σ.

Theorem 7 (Safety and Transparency).

1. **(Local Safety)** $\models [P]_\alpha \mid \mathsf{M} : \alpha : \langle \Gamma; \Delta \rangle$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$.
2. **(Local Transparency)** If $\models [P]_\alpha : \alpha : \langle \Gamma; \Delta \rangle$, then $[P]_\alpha \approx ([P]_\alpha \mid \mathsf{M})$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$.
3. **(Global Safety)** If N is fully monitored w.r.t. Σ, then $\models \mathsf{N} : \Sigma$.
4. **(Global Transparency)** Assum N and N have the same global transport $\langle r \ ; \ h \rangle$. If N is fully monitored w.r.t. Σ and $N = M \mid \langle r \ ; \ h \rangle$ is unmonitored but $\models M : \Sigma$, then we have $\mathsf{N} \sim N$.

Local safety (7.1) states that a monitored process always behaves well with respect to the specification. Local transparency (7.2) states that a monitored process behaves as an unmonitored process when the latter is well-behaved (e.g., it is statically checked). Global safety (7.3) states that a fully monitored network behaves well with respect to the given global specification. This property is closely related to session fidelity, introduced later in Theorem 11. Global transparency (7.4) states that a monitored network and an unmonitored network have equivalent behaviour when the latter is well-behaved with respect to the same (collection of) specifications.

By Proposition 4 and (7.2), we derive Corollary 8 stating that weakly bisimilar static networks combined with the same global transport are congruent.

Corollary 8 (Local transparency). If $\models [P]_\alpha \ : \ \alpha : \langle \Gamma; \Delta \rangle$, then for any $\langle r \ ; \ h \rangle$, we have $([P]_\alpha \mid \langle r \ ; \ h \rangle) \cong ([P]_\alpha \mid \mathsf{M} \mid \langle r \ ; \ h \rangle)$ with $\mathsf{M} = \alpha : \langle \Gamma; \Delta \rangle$.

By Theorem 7, we can *mix* unmonitored principals with monitored principals still obtaining the desired safety assurances.

In the following, we refer to a pair $\Sigma; \langle r \ ; \ h \rangle$ of a specification and a global transport as a *configuration*. The labelled transition relation for configurations, denoted by $\xrightarrow{\ell}_{\mathrm{g}}$, is relegated to [2]. Here it is sufficient to notice that the transitions of a configuration define the correct behaviours (with respect to Σ) in terms of the observation of inputs and outputs from/to the global transport $\langle r \ ; \ h \rangle$. We write that a configuration $\Sigma; \langle r \ ; \ h \rangle$ is *configurationally consistent* if all of its multi-step input transition derivatives are receivable and the resulting specifications Σ is consistent.

We also use $\xrightarrow{\ell}_{\mathrm{g}}$ to model globally visible transitions of networks (i.e., those locally visible transitions of a network that can be observed by its global transport). Below, we state that a message emitted by a valid output action is always receivable.

Lemma 9. Assume a network $N \equiv M | \langle r \ ; \ h \rangle$ conforming to $\Sigma; \langle r \ ; \ h \rangle$ which is configurationally consistent, if $N \xrightarrow{\ell}_{\mathrm{g}} N'$ such that ℓ is an output and $\Sigma; \langle r \ ; \ h \rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r \ ; \ h \cdot m \rangle$ then $h \cdot m$ is receivable to Σ'.

Also, we state that, as $N \equiv M \mid H$ and $\models M \ : \ \Sigma$, the satisfaction relation of M and Σ is preserved by transitions.

Lemma 10. Assume $N \equiv M \mid H$ and $\models M \ : \ \Sigma$. If $N \xrightarrow{\ell}_{\mathrm{g}} N' \equiv M' \mid H'$ and $\Sigma \xrightarrow{\ell} \Sigma'$, then $\models M' \ : \ \Sigma'$.

Theorem 11 (Session Fidelity). Assume configuration $\Sigma; \langle r \ ; \ h \rangle$ is configurationally consistent, and network $N \equiv M | \langle r \ ; \ h \rangle$ conforms to configuration $\Sigma; \langle r \ ; \ h \rangle$. For any ℓ, whenever we have $N \xrightarrow{\ell}_{\mathrm{g}} N'$ s.t. $\Sigma; \langle r \ ; \ h \rangle \xrightarrow{\ell}_{\mathrm{g}} \Sigma'; \langle r' \ ; \ h' \rangle$, it holds that $\Sigma'; \langle r' \ ; \ h' \rangle$ is configurationally consistent and N' conforms to $\Sigma'; \langle r' \ ; \ h' \rangle$.

By session fidelity, if all session message exchanges in a monitored/unmonitored network behave well with respect to the specifications (as communications occur), then this network exactly follows the original global specifications.

4 Conclusion and Future Work

We proposed a new formal safety assurance framework to specify and enforce the global safety for distributed systems, through the use of static and dynamic verification. We formally proved the correctness (with respect to distributed principals) of our architectural framework through a π-calculus based theory, identified in two key properties of dynamic network: global transparency and safety. We introduced a behavioural theory over monitored networks which allows compositional reasoning over trusted and untrusted (but monitored) components.

Implementation As a part of our collaboration with the Ocean Observatories Initiative [17], our theoretical framework is currently realised by an implementation, in which each monitor supports all well-formed protocols and is automatically self-configured, via session initiation messages, for all sessions that the endpoint participates in. Our implementation of the framework automates distributed monitoring by generating FSM from the local protocol projections. In this implementation, the global protocol serves as the key abstraction that helps unify the aspects of specification, implementation and verification (both static and dynamic) of distributed application development. Our experience has shown that the specification framework can accommodate diverse practical use cases, including real-world communication patterns used in the distributed services of the OOI cyberinfrastructure [17].

Future work Our objectives include the incorporation in the implementation of more elaborate handling of error cases into monitor functionality, such as halting all local sessions or coercing to valid actions [18, 16]. In order to reach this goal, we need to combine a simplification of [5] and nested sessions [10] to handle an exception inside MPSTs. We aim to construct a simple and reliable way to raise and catch exceptions in asynchronous networks. Our work is motivated by ongoing collaborations with the Savara[2] and Scribble[3] projects and OOI [17]. We are continuing the development of Scribble, its toolsuite and associated environments towards a full integration of sessions into the OOI infrastructure.

4.1 Related Work

Our work features a located, distributed process calculus to model monitored networks. Due to space limitations, we focus on the key differences with related work on dynamic monitoring.

The work in [13] proposes an ambient-based run-time monitoring formalism, called guardians, targeted at access control rights for network processes, and Klaim [9] advocates a hybrid (dynamic and static) approach for access control against capabilities (policies) to support static checking integrated within a dynamic access-control procedure. These works address specific forms of access control for mobility, while our more general approach aims at ensuring correct behaviour in sessions through a combination of static or run-time verification.

The work in [4] presents a monitor-based information-flow analysis in multiparty sessions. The monitors in [4] are inline (according to [6]) and control the information-flow by tagging each message with security levels. Our monitors are outline and aim at the application to distributed systems.

An informal approach to monitoring based on MPSTs, and an outline of monitors are presented in [8]. However, [8] only gives an overview of the desired properties, and requires all local processes to be dynamically verified through the protections of system monitors. In this paper, instead, we integrate statically

[2] `http://www.jboss.org/savara`
[3] `http://www.scribble.org`

and dynamically verified local processes into one network, and formally state the properties of this combination.

In summary, compared to these related works, our contribution focuses on the enforcement of global safety, with protocols specified as multiparty session types with assertions. It also provides formalisms and theorems for decentralised runtime monitoring, targeting interaction between components written in multiple (e.g., statically and dynamically typed) programming languages.

References

1. L. Bettini et al. Global progress in dynamically interleaved multiparty sessions. In *CONCUR*, volume 5201 of *LNCS*, pages 418–433, 2008.
2. L. Bocchi, T.-C. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. Technical Report 2013/3, Department of Computing, Imperial College London, 2013.
3. L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR*, volume 6269 of *LNCS*, pages 162–176, 2010.
4. S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini. Information flow safety in multiparty sessions. In *EXPRESS*, volume 64 of *EPTCS*, pages 16–30, 2011.
5. S. Capecchi, E. Giachino, and N. Yoshida. Global escape in multiparty session. In *FSTTCS'10*, volume 8 of *LIPICS*, pages 338–351, 2010.
6. F. Chen and G. Rosu. MOP:An Efficient and Generic Runtime Verification Framework. In *OOPSLA*, pages 569–588, 2007.
7. T.-C. Chen. *Theories for Session-based Governance for Large-Scale Distributed Systems*. PhD thesis, Queen Mary, University of London, 2013. To appear.
8. T.-C. Chen, L. Bocchi, P.-M. Deniélou, K. Honda, and N. Yoshida. Asynchronous distributed monitoring for multiparty session enforcement. In *TGC*, volume 7173 of *LNCS*, pages 25–45, 2011.
9. R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interaction and mobility. *IEEE Trans. Softw. Eng.*, 24:315–330, 1998.
10. R. Demangeon and K. Honda. Nested protocols in session types. In *CONCUR*, volume 7454 of *LNCS*, pages 272–286, 2012.
11. P.-M. Deniélou and N. Yoshida. Dynamic multirole session types. In *POPL*, pages 435–446, 2011.
12. P.-M. Deniélou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213, 2012.
13. G. Ferrari, E. Moggi, and R. Pugliese. Guardians for ambient-based monitoring. In *F-WAN*, pages 141–202. Elsevier, 2002.
14. K. Honda and N. Yoshida. On reduction-based process semantics. *TCS*, 151(2):437–486, 1995.
15. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL'08*, pages 273–284. ACM, 2008.
16. J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12:19:1–19:41, 2009.
17. OOI. `http://www.oceanobservatories.org/`.
18. F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3:30–50, 2000.
19. N. Yoshida, P.-M. Deniélou, A. Bejleri, and R. Hu. Parameterised multiparty session types. In *FoSSaCs'10*, volume 6014 of *LNCS*, pages 128–145, 2010.