Enforcing Availability in Failure-Aware Communicating Systems

López-Acosta, Hugo-Andrés; Nielson, Flemming; Nielson, Hanne Riis

Published in:
Proceedings of the 36th IFIP WG 6.1 International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2016)

Link to article, DOI:
10.1007/978-3-319-39570-8_13

Publication date:
2016

Document Version
Peer reviewed version

Enforcing Availability in Failure-Aware Communicating Systems

Hugo A. López, Flemming Nielson, and Hanne Riis Nielson

Technical University of Denmark
Kongens Lyngby, Denmark
{hulo,fnie,hrni}@dtu.dk

Abstract. Choreographic programming is a programming-language design approach that drives error-safe protocol development in distributed systems. Motivated by challenging scenarios in Cyber-Physical Systems (CPS), we study how choreographic programming can cater for dynamic infrastructures where the availability of components may change at runtime. We introduce the Global Quality Calculus (GC_q), a process calculus featuring novel operators for multiparty, partial and collective communications; we provide a type discipline that controls how partial communications refer only to available components; and we show that well-typed choreographies enjoy progress.

1 Introduction

Choreographies are a well-established formalism in concurrent programming, with the purpose of providing a *correct-by-construction* framework for distributed systems [8, 11]. Using Alice-Bob's style protocol narrations, they provide the structure of interactions among components in a distributed system. Combined with a behavioral type system, choreographies are capable of deriving distributed (endpoint) implementations. Endpoints generated from a choreography ascribe all and only the behaviors defined by it. Additionally, interactions among endpoints exhibit correctness properties, such as liveness and deadlock-freedom. In practice, choreographies guide the implementation of a system, either by automating the generation of correct deadlock-free code for each component involved, or by monitoring that the execution of a distributed system behaves according to a protocol [8, 34, 3].

In this paper we study the role of *availability* when building communication protocols. In short, availability describes the ability of a component to engage in a communication. Insofar, the study of communications using choreographies assumed that components were always available. We challenge this assumption on the light of new scenarios. The case of Cyber-Physical Systems (CPS) is one of them. In CPS, components become unavailable due to faults or because of changes in the environment. Even simple choreographies may fail when including availability considerations. Thus, a rigorous analysis of availability conditions in communication protocols becomes necessary, before studying more advanced properties, such as deadlock-freedom or protocol fidelity.

Practitioners in CPS take availability into consideration, programming applications in a *failure-aware* fashion. First, application-based QoS policies replace old node-based ones. Second, one-to-many and many-to-one communication patterns replace peer-to-peer communications. Still, programming a CPS from a component viewpoint such that it respects an application-based QoS is difficult, because there is no centralized way to ensure its enforcement.

This work reports initial steps towards a choreography-based approach for the development of failure-aware communication protocols, as exemplified by CPS. Our contribution is twofold. First, we present the Global Quality Calculus (GC_q), a process calculus aimed at capturing the most important aspects of CPS, such as variable availability conditions and multicast communications. It is a generalization of the Global Calculus [11], enriched with collective communication primitives and explicit availability considerations. Central to GC_q is the inclusion *quality predicates* [35] and optional datatypes, whose role is to allow for communications where only a subset of the original participants is available. Models in GC_q can accommodate in this way application-based QoS policies, instead of a node-centric approach.

Our second contribution relates to the verification of failure-aware protocols. We focus on *progress*. As an application-based QoS, a progress property requires that at least a minimum set of components is available before firing a communication action. Changing availability conditions may leave collective communications without enough required components, forbidding the completion of a protocol. We introduce a type system, orthogonal to session types, that ensures that well-typed protocols with variable availability conditions do not get stuck, preserving progress.

Document Structure In §2 we introduce the design considerations for a calculus with variable availability conditions and we present a minimal working example to illustrate the calculus in action. §3 introduces syntax and semantics of GC_q. The progress-enforcing type system is presented in §4. Section §5 discusses related work. Finally, §6 concludes. Appendix includes additional definitions and proof sketches of the main results, and is intended only for reviewing purposes.

2 Towards a Language for CPS Communications

The design of a language for CPS requires a *technology-driven* approach, that answers to requirements regarding the nature of communications and devices involved in CPS. Similar approaches have been successfully used for Web-Services [10, 38, 33], and Multicore Programming [28, 13]. The considerations on CPS used in this work come from well-established sources [2, 37]. We will proceed by describing their main differences with respect to traditional networks.

2.1 Unique Features in CPS Communications

Before defining a language for communication protocols in CPS, it is important to understand the taxonomy of networks where they operate. CPS are

composed by *sensor networks* (SN) that perceive important measures of a system, and *actuator networks* that change it. Some of the most important characteristics in these networks include asynchronous operation, sensor mobility, energy-awareness, application-based protocol fidelity, data-centric protocol development, and multicast communication patterns. We will discuss each of them.

Asynchrony. Depending on the application, deployed sensors in a network have less accessible mobile access points, for instance, sensors deployed in harsh environmental conditions, such as arctic or marine networks. Environment may also affect the lifespan of a sensor, or increase its probability of failure. To maximize the lifespan of some sensors, one might expect an *asynchronous operation*, letting sensors remain in a standby state, collecting data periodically.

Sensor Mobility. The implementation of sensors in autonomic devices brings about important considerations on *mobility*. A sensor can move away from the base station, making their interactions energy-intensive. In contrast, it might be energy-savvy to start a new session with a different base station closer to the new location.

Energy-Awareness. Limited by finite energetic resources, SN must optimize their energy consumption, both from node and application perspectives. From a node-specific perspective, a node in a sensor network can optimize its life by turning parts of the node off, such as the RF receiver. From a application-specific perspective, a protocol can optimize it energy usage by reducing its traffic. SN cover areas with dense node deployment, thus it is unnecessary that all nodes are operational to guarantee coverage. Additionally, SN must provide *self-configuration* capabilities, adapting its behavior to changing availability conditions. Finally, it is expected that some of the nodes deployed become permanently unavailable, as energetic resources ran out. It might be more expensive to recharge the nodes than to deploy new ones. The SN must be ready to cope with a decrease in some of the available nodes.

Data-Centric Protocols. One of the most striking differences to traditional networks is the *collaborative* behavior expected in SN. Nodes aim at accomplishing a similar, universal goal, typically related to maintaining an application-level quality of service (QoS). Protocols are thus data-centric rather than node-centric. Moreover, decisions in SN are made from the aggregate data from sensing nodes, rather than the specific data of any of them [36]. Collective decision-making based in aggregates is common in SN, for instance, in protocols suites such as SPIN [19] and Directed Diffusion [25]. Shifting from node-level to application-level QoS implies that *node fairness* is considerably less important than in traditional networks. In consequence, the analysis of *protocol fidelity* [22] requires a shift from node-based guarantees towards application-based ones.

Multicast Communication. Rather than peer-to-peer message passing, one-to-many and many-to-one communications are better solutions for energy-efficient

```
1 t1[S]{Acc1}, t2[S]{Acc2}, t3[S]{Acc3} start t0[M]{Acc0} : temperature(k);
2 t0{Acc0; Ms0} -> &q1 (t1{Acc1; Ms1}, t2{Acc2; Ms2}, t3{Acc3; Ms3}) : k[measure];
3 &q2 (t1{Ms1; E1}."1", t2{Ms2; E2}."−2", t3{Ms3; E3}."5") -> t0{Ms0; E0} : xm : ⟨k, avg⟩; 0
```

Fig. 1: Example: Sensor network choreography

SN, as reported in [18, 14]. However, as the number of sensor nodes in a SN scales to large numbers, communications between a base and sensing nodes can become a limiting factor. Many-to-one traffic patterns can be combined with data aggregation services (e.g.: TAG [30] or TinyDB [31]), minimizing the amount and the size of messages between nodes.

2.2 Model Preview

We will illustrate how the requirements for CPS communications have been assembled in the our calculus through a minimal example in Sensor Networks (SN). The syntax of our language is inspired on the Global Calculus [8, 11] extended with collective communication operations [28].

Example 1. Figure 1 portrays a simple SN choreography for temperature measurement. Line 1 models a *session establishment* phase between sensors $\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3$ (each of them implementing role S) and a monitor $\mathtt{t}_m$ with role M. In Line 2, $\mathtt{t}_m$ executes a *collective selection* of method *measure* at each node. In Line 3, an asynchronous many-to-one communication (e.g. *reduce*) is performed between sensors and the monitor. Quality predicates $\mathbf{q_1}, \mathbf{q_2}$ model application-based QoS, established in terms of availability requirements for each of the nodes. For instance, $\mathbf{q_1} = \mathbf{q_2} = \forall$ only allows communications with all sensors in place, and $\mathbf{q_1} = \forall, \mathbf{q_2} = 2/3$ tolerates the absence of one of the sensors in data harvesting. Once nodes satisfy applications' QoS requirements, an aggregation operation will be applied to the messages received, in this case computing the average value.

Considerations regarding the impact of available components in a communication must be tracked explicitly. Annotations $\{X; Y\}$ (in blue font) define *capabilities*, that is, control points achieved in the system. The X in $\mathtt{t}\{X; Y\}$ denotes the *required* capability for $\mathtt{t}$ to act, and Y describes the capability *offered* after $\mathtt{t}$ has engaged in an interaction. No preconditions are necessary for establishing a new session, so no required capabilities are necessary in Line 1. After a session has been established, capabilities $(Acc_i)_{i \in \{0...3\}}$ are available in the system. Lines 2 and 3 will modify which capabilities are present in the system depending on the number of available threads. For example, a model with $\mathbf{q_1} = 2/3$ can advance from $Acc_0, Acc_1, Acc_2, Acc_3$ to Ms_0, Acc_1, Ms_2, Ms_3. There may be cases in which an execution of the protocol will not generate necessary capabilities for a communication operation to be engaged, leaving a protocol stuck. One case will be if $\mathbf{q_1} = 2/3, \mathbf{q_2} = \forall$, since not enough Ms_i capabilities can be provided. We will defer the discussion about the interplay of capabilities and quality predicates to Section 4.

3 The Global Quality Calculus (GC_q)

In the following, C denotes a choreography; p denotes an annotated thread $\mathtt{t}[A]\{X;Y\}$, where $\mathtt{t}$ is a thread, X, Y are atomic formulae and A is a role annotation. We will use $\widetilde{\mathtt{t}}$ to denote $\{\mathtt{t}_1, \ldots, \mathtt{t}_j\}$ for a finite j. Variable a ranges over *service channels*, intuitively denoting the public identifier of a service, and $k \in \mathbf{N}$ ranges over a finite, countable set of session (names), created at runtime. Variable x ranges over variables local to a thread. We use terms t to denote data and expressions e to denote optional data, much like the use of option data types in programming languages like Standard ML [17]. Expressions include arithmetic and other first-order expressions excluding service and session channels. In particular, the expression $\mathsf{some}(t)$ signals the presence of some data t and none the absence of data. In our model, terms denote closed values v. Names m, n range over threads and session channels. Finally, $\mathbf{q}$ stands for a *quality predicate*, that determines when sufficient inputs/outputs are available. As an example, q can be $\exists$, meaning that one sender/receiver is required in the interaction, or it can be $\forall$ meaning that all of them are needed. We require $\mathbf{q}$ to be monotonic and satisfiable.

Definition 1 (GC_q syntax).

(Choreographies)	$C ::=$	$\eta;\, C \mid C + C \mid e@p?\, C : C \mid \mathbf{0}$	
(Annotated threads)	$p ::=$	$\mathtt{t}[A]\{X;Y\}$	
(Interactions)	$\eta ::=$	$\widetilde{p_r}$ **start** $\widetilde{p_s} : a(k)$	(*init*)
		$\mid\ p_r.e \mathrel{->} \&_{\mathbf{q}}(\widetilde{p_s : x_s}) : k$	(*broadcast*)
		$\mid\ \&_{\mathbf{q}}(\widetilde{p_r.e_r}) \mathrel{->} p_s : x : \langle k, op\rangle$	(*reduce*)
		$\mid\ p_r \mathrel{->} \&_{\mathbf{q}}(\widetilde{p_s}) : k[l]$	(*select*)

For simplicity of presentation, all models in the paper are finite. We will focus our discussion on the novel interactions. First, **start** defines a (multiparty) *session initiation* between active annotated threads $\widetilde{p_r}$ and annotated service threads $\widetilde{p_s}$. Each active thread (resp. service thread) implements the behaviour of one of the roles in $\widetilde{A_r}$ (resp. $\widetilde{A_s}$), sharing a new session name k. We assume that a session is established with at least two participating processes, therefore $2 \leq |\widetilde{p_r}| + |\widetilde{p_s}|$, and that threads in $\widetilde{p_r} \cup \widetilde{p_s}$ are pairwise different.

The language features broadcast, reduce and selection as collective interactions. A *broadcast* implements a one-to-many communication pattern, where a session channel k is used to transfer the evaluation of expression e (located at p_r) to threads in $\widetilde{p_s}$, with the resulting binding of variable x_i at p_i, for each $p_i \in \widetilde{p_s}$. A *reduce* combines one-to-many communications and aggregation [30]. In $\&_{\mathbf{q}}(\widetilde{p_r.e_r})$ -> $p_s : x : \langle k, op\rangle$, each annotated thread p_i in $\widetilde{p_r}$ evaluates an expression e_i, and the aggregate of all receptions is evaluated using op (an operator defined on multisets such as $\mathsf{max}, \mathsf{min}$, etc.) Interaction p_r -> $\&_{\mathbf{q}}(\widetilde{p_s}) : k[l]$ describes a collective *label selection*: p_r communicates the selection of label l to peers in $\widetilde{p_s}$ through session k.

Central to our language are *progress capabilities*. Pairs of atomic formulae $\{X; Y\}$ at each annotated thread state the necessary preconditions for a thread to engage (X), and the capabilities provided after its interaction (Y). As we will see in the semantics, there are no associated preconditions for session initiation (i.e. threads are created at runtime), so we normally omit them. Explicit $x@p/e@p$ indicate the variable/boolean expression x/e is located at p. We often omit $\mathbf{0}$, empty vectors and atomic formulae $\{X; Y\}$ from annotated threads when unnecessary.

The free term variables $\mathsf{fv}(C)$ are defined as usual. An interaction η in η**;** C can bind session channels, choreographies and variables. In **start**, variables $\{\widetilde{\rho_r}, a\}$ are free while variables $\{\widetilde{\rho_s}, k\}$ are bound (since they are freshly created). In broadcast, variables $\widetilde{x_s}$ are bound. A reduce binds $\{x\}$. Finally, we assume that all bound variables in an expression have been renamed apart from each other, and apart from any other free variables in the expression.

Expressivity The importance of roles is only crucial in a **start** interaction. Technically, one can infer the role of a given thread $\mathtt{t}$ used in an interaction η by looking at the **start** interactions preceding it in the abstract syntax tree. GC_q can still represent unicast message-passing patterns as in [8]. Unicast communication $p_1.e \text{ -> } p_2 : x : k$ can be encoded in multiple ways using broadcast/reduce operators. For instance, $p_1.e \text{ -> } \&_{\forall}(p_2 : x) : k$ and $\&_{\forall}(p_1.e) \text{ -> } p_2 : x : \langle id, k\rangle$ are just a couple of possible implementations.

3.1 Semantics

Choreographies are considered modulo standard structural and swapping congruence relations (resp. $\equiv$, $\simeq_C$). $\equiv$ is defined as the least congruence relation on C supporting $\alpha-$renaming, such that $(C, \mathbf{0}, +)$ is an abelian monoid. The swap congruence [11] provides a way to reorder non-conflicting interactions, allowing for a restricted form of asynchronous behavior. Non-conflicting interactions are those involving sender-receiver actions that do not conform a control-flow dependency. For instance, $\mathtt{t}_A.e_A \text{ -> } \&_{\mathbf{q_1}}(\mathtt{t}_B : x_B) : k_1$**;** $\mathtt{t}_C.e_C \text{ -> } \&_{\mathbf{q_2}}(\mathtt{t}_D : x_D) : k_2 \simeq_C \mathtt{t}_C.e_C \text{ -> } \&_{\mathbf{q_2}}(\mathtt{t}_D : x_D) : k_2$**;** $\mathtt{t}_A.e_A \text{ -> } \&_{\mathbf{q_1}}(\mathtt{t}_B : x_B) : k_1$. Formally, let $\mathbf{T}(C)$ be the set of threads in C, defined inductively as $\mathbf{T}(\eta; C) \stackrel{\mathtt{def}}{=} \mathbf{T}(\eta) \cup \mathbf{T}(C)$, and $\mathbf{T}(\eta) \stackrel{\mathtt{def}}{=} \bigcup_{i=\{1..j\}} \mathtt{t}_i$ if $\eta = \mathtt{t}_1[A_1].e \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_2[A_2] : x_2, \ldots, \mathtt{t}_j[A_j] : x_j) : k$ (similarly for init, reduce and selection, and standardly for the other process constructs in C). The swapping congruence rules are presented in Figure 2.

A state σ keeps track of the capabilities achieved by a thread in a session, and it is formally defined as set of maps $(\mathtt{t}, k) \mapsto X$. The rules in Figure 3 define state manipulation operations, including update ($\sigma[\sigma']$), and lookup ($\sigma(\mathtt{t}, k)$).

Because of the introduction of quality predicates, a move from η**;** C into C might leave some variables in η without proper values, as the participants involved might not have been available. We draw inspiration from [35], introducing *effect* rules describing how the evaluation of an expression in a reduce

$$\frac{\mathbf{T}(\eta) \# \mathbf{T}(\eta')}{\eta;\ (\eta';\ C) \simeq_C \eta';\ (\eta;\ C)} \qquad \frac{p \notin \mathbf{T}(\eta)}{e@p?\,\eta;\ C_1 \ :\ \eta;\ C_2 \simeq_C \eta;\ e@p?\,C_1 \ :\ C_2}$$

$$\frac{p \neq r}{\begin{array}{l} e@p?\,(e'@r?\,C_1 \ :\ C_2) \ : \\ \qquad (e'@r?\,C_1' \ :\ C_2') \end{array} \simeq_C \begin{array}{l} e'@r?\,(e@p?\,C_1 \ :\ C_1') \ : \\ \qquad (e@p?\,C_2 \ :\ C_2') \end{array}}$$

Fig. 2: Swap congruence relation, $\simeq_C$

$$\frac{Y = X \text{ if } (\mathfrak{t}, k, X) \in \sigma \quad Y = \emptyset \quad \text{o.w.}}{\sigma(\mathfrak{t}, k) = Y} \qquad \frac{\delta = \{(\mathfrak{t}, k, X) \mid (\mathfrak{t}, k, X) \in \sigma \wedge (\mathfrak{t}, k, Y) \in \sigma'\}}{\sigma[\sigma'] = (\sigma \backslash \delta), \sigma'}$$

Fig. 3: State lookup and update rules

operation affects interactions. The relation $\twoheadrightarrow$ (Figure 4) describes how evaluations are partially applied without affecting waiting threads. Label ξ records the substitutions of atomic formulae in each thread.

Finally, given $\phi \in \{\mathtt{tt}, \mathtt{ff}\}$, the relation $\beta ::_\phi \theta$ tracks whether all required binders in β have been performed, as well as substitutions used θ. Binder β is defined in terms of partially evaluated outputs c:

$$sc ::= \quad p.e \quad \mid \quad p.\mathsf{some}(v) \qquad\qquad c ::= \quad \&_q(sc_1, \ldots, sc_n)$$

The rules specifying $\beta ::_\phi \theta$ appear in Figure 5. A substitution $\theta = [(p_1, \mathsf{some}(v_1)), \ldots, (p_n, \mathsf{some}(v_n))/x_1@p_1, \ldots, x_n@p_n]$ maps each variable x_i at p_i to optional data $\mathsf{some}(v_i)$ for $1 \leq i \leq n$. A composition $\theta_1 \circ \theta_2(x)$ is defined as $\theta_1 \circ \theta_2(x) ::= \theta_1(\theta_2(x))$, and $q(t_1, \ldots, t_n) = \bigwedge_{i \in 1 \leq i \leq n} t_i$ if $q = \forall$, $q(t_1, \ldots, t_n) = \bigvee_{i \in 1 \leq i \leq n} t_i$ if $q = \exists$, and possible combinations therein. As for process terms, $\theta(C)$ denotes the application of substitution θ to a term C (and similarly for η).

We now have all the ingredients to understand the semantics of GC_q. The set of transition rules in $\xrightarrow{\lambda}$ is defined as the minimum relation on names, states, and choreographies satisfying the rules in Figure 6. The operational semantics is given in terms of labelled transition rules. Intuitively, a transition $(\nu\widetilde{m})\ \langle \sigma, C \rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C' \rangle$ expresses that a configuration $\langle \sigma, C \rangle$ with used names $\widetilde{m}$ fires an action λ and evolves into $\langle \sigma', C' \rangle$ with names $\widetilde{n}$. We use the shorthand notation $A \# B$ to denote set disjointness, $A \cap B = \emptyset$. The exchange function $[\![X; Y]\!]Z$ returns $(Z \backslash X) \cup Y$ if $X \subseteq Z$ and Z otherwise. Actions are defined as $\lambda ::= \{\tau, \eta\}$, where η denotes interactions, and τ represents an internal computation. Relation $e@p \downarrow v$ describes the evaluation of a expression e (in p) to a value v.

We now give intuitions on the most representative operational rules. Rule [Init] models initial interactions: state σ is updated to account for the new threads in the session, updating the set of used names in the reductum. Rule [Bcast] models broadcast: given an expression evaluated at the sender, one needs to check

$$\frac{\eta = \&_{\mathbf{q}}(\mathtt{t}_1[A_1]\{X_1;Y_1\}.e_1,\ldots,\mathtt{t}_j[A_j]\{X_j;Y_j\}.e_j) \twoheadrightarrow \mathtt{t}_B[B]\{X_B;Y_B\} : x : \langle k, op\rangle \quad e_i@\mathtt{t}_i \downarrow v_i \quad X_i \in \sigma(\mathtt{t}_i,k) \quad \sigma' = \sigma[(\mathtt{t}_i,k) \mapsto [\![X_i;Y_i]\!](\sigma(\mathtt{t}_i,k))] \quad {\scriptstyle i\in\{1\ldots j\}}}{\langle \sigma, \eta;\, C\rangle \xrightarrow{(\mathtt{t}_i,k):\, X_i :: Y_i} \left\langle \sigma',\ (\&_{\mathbf{q}}(\mathtt{t}_1[A_1]\{X_1;Y_1\}.e_1,\ldots,\mathtt{t}_i[A_i]\{Y_i;Y_i\}.\mathsf{some}(v_i),\ldots,\mathtt{t}_j[A_j]\{X_j;Y_j\}.e_j) \twoheadrightarrow \mathtt{t}_B[B]\{X_B;Y_B\} : x : \langle k, op\rangle)\ ;\ C \right\rangle}$$

Fig. 4: Effects

$$\frac{}{p.e ::_{\mathtt{ff}} [\,]} \qquad \frac{}{p.\mathsf{some}(v) ::_{\mathtt{tt}} [(p, \mathsf{some}(v))]} \qquad \frac{sc_1 ::_{t_1} \theta_1 \quad \ldots \quad sc_n ::_{t_n} \theta_n}{\&_q(sc_1,\ldots,sc_n) ::_{q(t_1,\ldots,t_n)} \theta_1 \circ \ldots \circ \theta_n}$$

Fig. 5: Rules for $\beta ::_\phi \theta$

that there are enough receivers ready to get a message. Such a check is performed by evaluating $q(J)$. In case of a positive evaluation, the execution of the rule will: (1) update the current state with the new states of each participant engaged in the broadcast, and (2) apply the partial substitution θ to the continuation C. The behaviour of a reduce operation is described using rules ⌊RedD⌉ and ⌊RedE⌉: the evaluation of expressions of each of the available senders generates an application of the effect rule in Figure 4. If all required substitutions have been performed, one can proceed by evaluating the operator to the set of received values, binding variable x to its results, otherwise the choreography will wait until further inputs are received (i.e.: the continuation is delayed).

In contrast to previous works in multiparty sessions (e.g. [12]), we present an *early* semantics: it allows for transitions to match with distinct moves, depending on which participants are available first. We opted to favor an application-based QoS rather than a node-based QoS, as described in Section 2. Similar considerations motivates the asymmetry between broadcast and reduce: while a broadcast is a *non-blocking* operation that fires as long as enough receivers are ready to be engaged, a reduce is a *blocking* operation, and will delay the transition until there is enough senders.

The reader familiar with the Global Calculus may have noticed the absence of a general asynchronous behaviour in our setting. In particular, rule:

$$\frac{(\nu\widetilde{m})\ \langle\sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle\sigma', C'\rangle \quad \eta \neq \mathbf{start} \quad snd(\eta) \subseteq \mathsf{fn}(\lambda) \quad rcv(\eta)\ \#\ \mathsf{fn}(\lambda) \quad \widetilde{n} = \widetilde{m}, \widetilde{r} \quad \forall_{r\in\widetilde{r}}\ (r \in \mathsf{bn}(\lambda) \quad r \notin \mathsf{fn}(\eta))}{(\nu\widetilde{m})\ \langle\sigma, \eta;\, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle\sigma', \eta;\, C'\rangle}\ \lfloor\mathsf{Asynch}\rceil$$

corresponding to the extension of rule $\lfloor^{\mathrm{C}}|_{\mathrm{ASYNCH}}\rceil$ in [11] with collective communications, is absent in our semantics. The reason behind it lies in the energy considerations of our application: consecutive communications may have different energetic costs, affecting the availability of sender nodes. Consider for example the configuration

$$(\nu\widetilde{m})\ \langle\sigma, (\mathtt{t}_A[A]\{X;Y\}.e \twoheadrightarrow \&_\exists(\widetilde{\mathtt{t}_r[B_r] : x_r}) : k);\ \mathtt{t}_A[A]\{X;Y\}.e \twoheadrightarrow \&_\forall(\widetilde{\mathtt{t}_s[B_s] : x_s}) : k\rangle$$

$$\frac{\eta = \mathtt{t}_r[A_r]\{\widetilde{X_r; Y_r}\}\ \mathbf{start}\ \mathtt{t}_s[B_s]\{\widetilde{Y_s}\} : a(k) \quad \sigma' = [(\mathtt{t}_i, k) \mapsto Y_i]_{i=1}^{|\widetilde{\mathtt{t}_r}|+|\widetilde{\mathtt{t}_s}|} \quad \widetilde{n} = \widetilde{\mathtt{t}_s}, \{k\} \quad \widetilde{n} \,\#\, \widetilde{m}}{(\nu\widetilde{m})\ \left\langle \sigma, \mathtt{t}_r[\widetilde{A_r]\{Y_r}\}\ \mathbf{start}\ \mathtt{t}_s[\widetilde{B_s]\{Y_s}\} : a(k); C \right\rangle \xrightarrow{\eta} (\nu\widetilde{m}, \widetilde{n})\ \langle \sigma[\sigma'], C\rangle}\ \lfloor \text{Init} \rceil$$

$$\frac{\eta = \mathtt{t}_A[A]\{X_A; Y_A\}.e \mathrel{->} \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\} : x_r) : k \quad J \subseteq \widetilde{\mathtt{t}_r} \quad q(J) \quad e@\mathtt{t}_A \downarrow v \quad \forall_{i\in\{A\}\cup J} : X_i \subseteq \sigma(\mathtt{t}_i, k) \wedge \sigma'(t_i, k) = [\![X_i; Y_i]\!](\sigma(t_i, k)) \quad \forall_{i\in\widetilde{\mathtt{t}_r}} : \theta(x_i) = \begin{cases} \mathsf{some}(v) & i \in J \\ \mathsf{none} & \text{o.w.} \end{cases}}{(\nu\widetilde{m})\ \left\langle \sigma, \left(\mathtt{t}_A[A]\{X_A; Y_A\}.e \mathrel{->} \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\} : x_r) : k\right); C \right\rangle \xrightarrow{\theta(\eta)} (\nu\widetilde{m})\ \langle \sigma[\sigma'], \theta(C)\rangle}\ \lfloor \text{Bcast} \rceil$$

$$\frac{\eta = \mathtt{t}_A[A]\{X_A; Y_A\} \mathrel{->} \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\}) : k[l_h] \quad J \subseteq \widetilde{\mathtt{t}_r} \quad q(J) \quad \forall_{i\in\{A\}\cup J} : X_i \subseteq \sigma(\mathtt{t}_i, k) \wedge \sigma'(\mathtt{t}_i, k) = [\![X_i; Y_i]\!](\sigma(\mathtt{t}_i, k))}{(\nu\widetilde{m})\ \left\langle \sigma, \left(\mathtt{t}_A[A]\{X_A; Y_A\} \mathrel{->} \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\}) : k[l_h]\right); C \right\rangle \xrightarrow{\eta} (\nu\widetilde{m})\ \langle \sigma[\sigma'], C\rangle}\ \lfloor \text{Sel} \rceil$$

$$\frac{\eta = \&_{\mathbf{q}}(\mathtt{t}_r[A_r]\{\widetilde{X_r; Y_r}\}.e_r) \mathrel{->} \mathtt{t}_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle \quad \langle \sigma, \eta; C\rangle \xrightarrow{\xi} \langle \sigma', \eta'; C\rangle \quad \eta' ::_{\mathtt{ff}} \theta}{(\nu\widetilde{m})\ \langle \sigma, \eta; C\rangle \xrightarrow{\tau} (\nu\widetilde{m})\ \langle \sigma', \eta'; C\rangle}\ \lfloor \text{RedD} \rceil$$

$$\frac{\eta = \&_{\mathbf{q}}(\mathtt{t}_r[A_r]\{\widetilde{X_r; Y_r}\}.e_r) \mathrel{->} \mathtt{t}_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle \quad \langle \sigma, \eta; C\rangle \xrightarrow{\xi} \langle \sigma', \eta'; C\rangle \quad \eta' ::_{\mathtt{tt}} \theta \quad (\mathtt{t}_B, k, X_B) \in \sigma'}{(\nu\widetilde{m})\ \langle \sigma, \eta; C\rangle \xrightarrow{\theta(\eta')} (\nu\widetilde{m})\ \langle [\![(\mathtt{t}_B, k, X_B); (\mathtt{t}_B, k, Y_B)]\!]\sigma', C[\mathsf{op}(\theta)/x@\mathtt{t}_B]\rangle}\ \lfloor \text{RedE} \rceil$$

$$\frac{C\,\mathcal{R}\,C' \quad (\nu\widetilde{m})\ \langle \sigma, C'\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C''\rangle \quad C''\,\mathcal{R}\,C''' \quad \mathcal{R} \in \{\equiv, \simeq_C\}}{(\nu\widetilde{m})\ \langle \sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'''\rangle}\ \lfloor \text{Cong} \rceil$$

$$\frac{i = 1 \text{ if } e@\mathtt{t} \downarrow \mathtt{tt}, \quad i = 2 \ \text{ o.w.}}{(\nu\widetilde{m})\ \langle \sigma, e@\mathtt{t}?\, C_1 \,:\, C_2\rangle \xrightarrow{\tau} (\nu\widetilde{m})\ \langle \sigma, C_i\rangle}\ \lfloor \text{If} \rceil \qquad \frac{(\nu\widetilde{m})\ \langle \sigma, C_i\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'\rangle \quad i\in\{1,2\}}{(\nu\widetilde{m})\ \langle \sigma, C_1 + C_2\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'\rangle}\ \lfloor \text{Sum} \rceil$$

Fig. 6: GC_q: Operational Semantics

with $\widetilde{\mathtt{t}_r} \# \widetilde{\mathtt{t}_s}$ and $X \subseteq \sigma(\mathtt{t}_A, k)$. If the order of the broadcasts is shuffled, the second broadcast may consume all energy resources for $\mathtt{t}_A$, making it unavailable later. Formally, the execution of a broadcast update the capabilities offered in σ for $\mathtt{t}_A, k$ to Y, inhibiting two communication actions with same capabilities to be reordered. We will refrain the use Rule $\lfloor$Asynch$\rceil$ in our semantics.

Definition 2 (Progress). *C progresses if there exists $C', \sigma', \widetilde{n}, \lambda$ such that $(\nu\widetilde{m})\ \langle \sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'\rangle$, for all $\sigma, \widetilde{m}$.*

4 Type-checking Progress

One of the challenges regarding the use of partial collective operations concerns the possibility of getting into runs with locking states. Consider a variant of

Example 1 with $\mathbf{q_1} = \exists$ and $\mathbf{q_2} = \forall$. This choice leads to a blocked configuration. The system blocks since the collective selection in Line (2) continues after a subset of the receivers in $\mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3$, have executed the command. Line (3) requires all senders to be ready, which will not be the most general case. The system will additionally block if participant dependencies among communications is not preserved. The choreography in Figure 7 illustrates this.

```
2  ... Lines 1,2 in Figure 1.
3  &∃(t1{Ms1; E1}."1", t3{Ms3; E3}."5") -> tm{Ms0; E0} : x0 : ⟨k, avg⟩; 0
```

Fig. 7: Variant of Example 1 with locking states

The choreography in Figure 7 blocks for $\mathbf{q_1} = \exists$, since the selection operator in Line (2) can execute a communication over $\mathtt{t}_2$, blocking the reduce in Line 3.

We introduce a type system to ensure progress on variable availability conditions. A judgment is written as $\Psi \vdash C$, where Ψ is a list of formulae in Intuitionistic Linear Logic (ILL) [16]. Intuitively, $\Psi \vdash C$ is read as *the formulae in Ψ describe the program point immediately before C*. Formulae $\psi \in \Psi$ take the form of the constant $\mathtt{tt}$, ownership types of the form $p : k\,[A] \triangleright X$, and the linear logic version of conjunction, disjunction and implication $(\otimes, \oplus, \multimap)$. Here $p : k\,[A] \triangleright X$ is an *ownership type*, asserting that p behaves as the role A in session k with atomic formula X. Moreover, we require Ψ to contain formulae free of linear implications in $\Psi \vdash C$[1].

Figure 8 presents selected rules for the type system for GC_q. The full definition is included in Appendix A.1. Since the rules for inaction, conditionals and non-determinism are standard, we focus our explanation on the typing rules for communications. Rule $\lfloor$TInit$\rceil$ types new sessions: Ψ is extended with function $\mathbf{init}(\mathtt{t}_p\widetilde{[A]\{X\}}, k)$, that returns a list of ownership types $\mathtt{t}_p : \widetilde{k\,[A] \triangleright X}$. The condition $\{\widetilde{\mathtt{t}_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))$ ensures that new names do not exist neither in the threads nor in the used keys in Ψ.

The typing rules for broadcast, reduce and selection are analogous, so we focus our explanation in $\lfloor$TBcast$\rceil$. Here we abuse of the notation, writing $\Psi \vdash C$ to denote type checking, and $\Psi \vdash \psi$ to denote formula entailment. The semantics of $\forall^{\geq 1} J$ s.t. $\mathbf{C} : D$ is given by $\forall J$ s.t. $\mathbf{C} : D \wedge \exists J$ s.t. $\mathbf{C}$. The judgment

$$\Psi \vdash (\mathtt{t}_A[A]\{X_A; Y_A\}.e \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\widetilde{\{X_r; Y_r\}} : x_r) : k); C$$

succeeds if environment Ψ can provide capabilities for sender $\mathtt{t}_A[A]$ and for a valid subset J of the receivers in $\widetilde{\mathtt{t}_r[B_r]}$. J is a valid subset if it contains enough threads to render the quality predicate true ($q(J)$), and the proof of $\psi_A, (\psi_j)_{j \in J}, \phi \multimap \phi' \vdash \phi'$ is provable. This proof succeeds if ψ_A and $(\psi_j)_{j \in J}$ contain ownership types for the sender and available receivers with corresponding

[1] We do, however, use the full set of operators when performing proof search

Choreography Formation $(\Psi \vdash C)$,

$$\frac{\Psi, \mathbf{init}(\mathtt{t}_r[\widetilde{A_r}]\{Y_r\}, \mathtt{t}_s[\widetilde{B_s}]\{Y_s\}, k) \vdash C \quad \{\widetilde{\mathtt{t}_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))}{\Psi \vdash \mathtt{t}_r[\widetilde{A_r}]\{Y_r\} \ \mathbf{start} \ \mathtt{t}_s[\widetilde{B_s}]\{Y_s\} : a(k); \ C} \lfloor \mathsf{Tinit} \rceil$$

$$\frac{\begin{array}{c}\forall^{\geq 1} J.\ s.t. \left(\begin{array}{c} J \subseteq \widetilde{\mathtt{t}_r} \ \wedge \ q(J) \ \wedge \ \Psi = \psi_A, (\psi_j)_{j\in J}, \Psi' \\ \wedge \ \psi_A, (\psi_j)_{j\in J} \vdash \mathtt{t}_A : k\,[A] \rhd X_A \bigotimes_{j\in J} (\mathtt{t}_j : k\,[B_j] \rhd X_j) \end{array}\right) : \\ \mathtt{t}_A : k\,[A] \rhd Y_A, (\mathtt{t}_j : k\,[B_j] \rhd Y_j)_{j\in J}, \Psi' \vdash C \quad \vdash e@\mathtt{t}_A : \mathbf{opt.data} \quad (\vdash x_i@\mathtt{t}_i : \mathbf{opt.data})_{i=1}^{|\widetilde{\mathtt{t}_r}|}\end{array}}{\Psi \vdash \left(\mathtt{t}_A[A]\{X_A; Y_A\}.e \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\} : x_r) : k\right); \ C} \lfloor \mathsf{Tbcast} \rceil$$

$$\frac{\begin{array}{c}\forall^{\geq 1} J.\ s.t. \left(\begin{array}{c} J \subseteq \widetilde{\mathtt{t}_r} \ \wedge \ q(J) \ \wedge \ \Psi = \psi_B, (\psi_j)_{j=1}^{|J|}, \Psi' \\ \wedge \ \psi_B, (\psi_j)_{j=1}^{|J|} \vdash \mathtt{t}_B : k\,[B] \rhd X_B \bigotimes_{j\in J} (\mathtt{t}_j : k\,[A_j] \rhd X_j) \end{array}\right) : \\ \mathtt{t}_B : k\,[B] \rhd Y_B, (\mathtt{t}_j : k\,[A_j] \rhd Y_j)_{j=1}^{|J|}, \Psi' \vdash C \quad (\vdash e_i@\mathtt{t}_i : \mathbf{opt.data})_{i=1}^{|\widetilde{\mathtt{t}_r}|} \quad \vdash x@\mathtt{t}_B : \mathbf{opt.data}\end{array}}{\Psi \vdash \left(\&_{\mathbf{q}}(\mathtt{t}_r[A_r]\widetilde{\{X_r; Y_r\}}.e_r) \text{ -> } \mathtt{t}_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle\right); \ C} \lfloor \mathsf{Tred} \rceil$$

$$\frac{((\text{as in } \lfloor \textsc{Tbcast} \rceil^*))}{\Psi \vdash \left(\mathtt{t}_A[A]\{X_A; Y_A\} \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\widetilde{\{X_r; Y_r\}}) : k[l_h]\right); \ C} \lfloor \mathsf{Tsel} \rceil$$

$$\frac{}{\Psi \vdash \mathbf{0}} \lfloor \mathsf{Tinact} \rceil \qquad \frac{\Psi \vdash C_1 \quad \Psi \vdash C_2}{\Psi \vdash e@\mathtt{t}?\, C_1 \ : \ C_2} \lfloor \mathsf{Tcond} \rceil \qquad \frac{\Psi = \psi \oplus \psi' \quad \psi \vdash C \quad \psi' \vdash C'}{\Psi \vdash C + C'} \lfloor \mathsf{Tsum} \rceil$$

Fig. 8: GC_q: Type checking rules (excerpt): Premises for $\lfloor$Tsel$\rceil$ are the same as for $\lfloor$Tbcast$\rceil$, without **opt.data** premises

capabilities. Finally, the type of the continuation C will consume the resources used in the sender and all involved receivers, updating them with new capabilities for the threads engaged.

Example 2. In Example 1, $\mathtt{tt} \vdash C$ if $(\mathbf{q_1} = \forall) \wedge (\mathbf{q_2} = \{\forall, \exists\})$. In the case $\mathbf{q_1} = \exists, \mathbf{q_2} = \forall$, the same typing fails. Similarly, $\mathtt{tt} \nvdash C$ if $\mathbf{q_1} = \exists$, for the variant of Example 1 in Figure 7.

A type preservation theorem must consider the interplay between the state and formulae in Ψ. We write $\sigma \models \Psi$ to say that the tuples in σ entail the formulae in Ψ. For instance, $\sigma \models \mathtt{t}: k\,[A] \rhd X$ iff $(\mathtt{t}, k, X) \in \sigma$. Its formal definition is included in Appendix A.1.

Theorem 1 (Type Preservation). *If* $(\nu\widetilde{m})\ \langle \sigma, C\rangle \xrightarrow{\lambda} (\nu\widetilde{n})\ \langle \sigma', C'\rangle$, $\sigma \models \Psi$, *and* $\Psi \vdash C$, *then* $\exists \Psi'.\ \Psi' \vdash C'$ *and* $\sigma' \models \Psi'$.

Theorem 2 (Progress). *If* $\Psi \vdash C$, $\sigma \models \Psi$ *and* $C \not\equiv \mathbf{0}$, *then* C *progresses.*

The decidability of type checking depends on the provability of formulae in our ILL fragment. Notice that the formulae used in type checking corresponds to the Multiplicative-Additive fragment of ILL, whose provability is decidable

[27]. For typing collective operations, the number of checks grows according to the amount of participants involved. Decidability exploits the fact that for each interaction the number of participants is bounded.

Theorem 3 (Decidability of Typing). *$\Psi \vdash C$ is decidable*

5 Related Work

Availability considerations in distributed systems has recently spawned novel research strands in regular languages [21, 1], continuous systems [2], and endpoint languages [35]. To the best of our knowledge, this is the first work considering availability from a choreographical perspective.

A closely related work is the Design-By-Contract approach for multiparty interactions [4]. In fact, in both works communication actions are enriched with pre-/post- conditions, similar to works in sequential programming [20]. The work on [4] enriches global types with assertions, that are then projected to a session $\pi-$calculus. Assertions may generate ill-specifications, and a check for consistency is necessary. Our capability-based type system guarantees temporal-satisfiability as in [4], not requiring history-sensitivity due to the simplicity of the preconditions used in our framework. The most obvious difference with [4] is the underlying semantics used for communication, that allows progress despite some participants are unavailable.

Other works have explored the behavior of communicating systems with collective/broadcast primitives. In [24], the expressivity of a calculus with bounded broadcast and collection is studied. In [28], the authors present a type theory to check whether models for multicore programming behave according to a protocol and do not deadlock. Our work differs from these approaches in that our model focuses considers explicit considerations on availability for the systems in consideration. Also for multicore programming, the work in [13] presents a calculus with fork/join communication primitives, with a flexible phaser mechanism that allows some threads to advance prior to synchronization. The type system guarantees a node-centric progress guarantee, ideal for multicore computing, but too coarse for CPS. Finally, the work [26], present endpoint (session) types for the verification of communications using broadcast in the Ψ-calculus. We do not observe similar considerations regarding availability of components in this work.

The work [12] presented multiparty global types with join and fork operators, capturing in this way some notions of broadcast and reduce communications, which is similar to our capability type-system. The difference with our approach is described in Section 3. On the same branch [15] introduces multiparty global types with recursion, fork, join and merge operations. The work does not provide a natural way of encoding broadcast communication, but one could expect to be able to encode it by composing fork and merge primitives.

6 Conclusions and Future Work

We have presented a process calculus aimed at studying protocols with variable availability conditions, as well as a type system to ensure their progress. It constitutes the first step towards a methodology for the safe development of communication protocols in CPS. The analysis presented is orthogonal to existing type systems for choreographies (c.f. session types [11].) Our next efforts include the modification of the type theory to cater for recursive behavior, the generation of distributed implementations (e.g. EndPoint Projection [8]), and considerations of compensating [7, 9, 29] and timed behavior [6, 5]. Type checking is computationally expensive, because for each collective interaction one must perform the analysis on each subset of participants involved. The situation will be critical once recursion is considered. We believe that the efficiency of type checking can be improved by modifying the theory so it generates one formulae for all subsets.

Traditional design mechanisms (including sequence charts of UML and choreographies) usually focus on the desired behavior of systems. In order to deal with the challenges from security and safety in CPS it becomes paramount to cater for failures and how to recover from them. This was the motivation behind the development of the Quality Calculus that not only extended a π-calculus with quality predicates and optional data types, but also with mechanisms for programming the continuation such that both desired and undesired behavior was adequately handled. In this work we have incorporated the quality predicates into choreographies and thereby facilitate dealing with systems in a failure-aware fashion. However, it remains a challenge to incorporate the consideration of both desired and undesired behavior that is less programming oriented (or EndPoint Projection oriented) than the solution presented by the Quality Calculus. This may require further extensions of the calculus with *fault-tolerance* considerations.

Acknowledgments. We would like to thank Marco Carbone and Jorge A. Pérez for their insightful discussions, and to all anonymous reviewers for their helpful comments improving the paper. This research was funded by the Danish Foundation for Basic Research, project *IDEA4CPS* (DNRF86-10). López has benefitted from travel support by the EU COST Action IC1201: *Behavioural Types for Reliable Large-Scale Software Systems* (BETTY).

References

1. P. A. Abdulla, M. F. Atig, R. Meyer, and M. S. Salehi. What's decidable about availability languages? In *FSTTCS*, volume 45 of *LIPIcs*, pages 192–205. Schloss Dagstuhl, 2015.
2. R. Alur. *Principles of Cyber-Physical Systems.* MIT Press, 2015.
3. L. Bocchi, T. Chen, R. Demangeon, K. Honda, and N. Yoshida. Monitoring networks through multiparty session types. In *FMOODS/FORTE*, volume 7892 of *LNCS*, pages 50–65. Springer, 2013.

4. 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. Springer, 2010.
5. L. Bocchi, J. Lange, and N. Yoshida. Meeting deadlines together. In *CONCUR*, volume 42 of *LIPIcs*, pages 283–296. Schloss Dagstuhl, 2015.
6. L. Bocchi, W. Yang, and N. Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.
7. M. Carbone. Session-based choreography with exceptions. *Electr. Notes Theor. Comput. Sci.*, 241:35–55, 2009.
8. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *ESOP*, pages 2–17, 2007.
9. M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions in session types. In *CONCUR*, volume 5201 of *LNCS*, pages 402–417. Springer, 2008.
10. M. Carbone, K. Honda, N. Yoshida, R. Milner, G. Brown, and S. Ross-Talbot. A theoretical basis of communication-centred concurrent programming. *Web Services Choreography Working Group mailing list, to appear as a WS-CDL working report*, 2006.
11. M. Carbone and F. Montesi. Deadlock-freedom-by-design: Multiparty asynchronous global programming. In *POPL*, pages 263–274. ACM, 2013.
12. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multiparty sessions. In *FORTE*, pages 1–28. Springer, 2011.
13. T. Cogumbreiro, F. Martins, and V. T. Vasconcelos. Coordinating phased activities while maintaining progress. In *COORDINATION*, volume 7890 of *LNCS*, pages 31–44. Springer, 2013.
14. J. Deng, Y. S. Han, W. B. Heinzelman, and P. K. Varshney. Balanced-energy sleep scheduling scheme for high-density cluster-based sensor networks. *Computer communications*, 28(14):1631–1642, 2005.
15. P. Denielou and N. Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211, pages 194–213. Springer, 2012.
16. J.-Y. Girard. Linear logic. *Theor. Comp. Sci.*, 50:1–102, 1987.
17. R. Harper. *Programming in Standard ML*. Working Draft, 2013.
18. W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660–670, 2002.
19. W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom*, pages 174–185. ACM, 1999.
20. C. A. R. Hoare. An axiomatic basis for computer programming (reprint). *Commun. ACM*, 26(1):53–56, 1983.
21. J. Hoenicke, R. Meyer, and E. Olderog. Kleene, Rabin, and Scott are available. In *CONCUR*, volume 6269 of *LNCS*, pages 462–477. Springer, 2010.
22. K. Honda, V. T. Vasconcelos, and M. Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP*, pages 122–138. Springer, 1998.
23. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. *POPL*, 43(1):273–284, 2008.
24. H. Hüttel and N. Pratas. Broadcast and aggregation in BBC. In *PLACES*, EPTCS, pages 51–62, 2015.
25. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *MobiCom*, pages 56–67. ACM, 2000.

26. D. Kouzapas, R. Gutkovas, and S. J. Gay. Session types for broadcasting. In *PLACES*, volume 155 of *EPTCS*, pages 25–31, 2014.
27. P. Lincoln. Deciding provability of linear logic formulas. *London Mathematical Society Lecture Note Series*, pages 109–122, 1995.
28. H. A. López, E. R. B. Marques, F. Martins, N. Ng, C. Santos, V. T. Vasconcelos, and N. Yoshida. Protocol-based verification of message-passing parallel programs. In *OOPSLA*, pages 280–298. ACM, 2015.
29. H. A. López and J. A. Pérez. Time and Exceptional Behavior in Multiparty Structured Communications. In *WS-FM*, volume 7176 of *LNCS*, pages 48–63. Springer, 2012.
30. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(SI):131–146, 2002.
31. S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD/PODS international conference on Management of data*, pages 491–502. ACM, 2003.
32. F. Montesi. *Choreographic Programming*. Ph.D. thesis, IT University of Copenhagen, 2013. `http://www.fabriziomontesi.com/files/m13_phdthesis.pdf`.
33. F. Montesi, C. Guidi, and G. Zavattaro. Composing services with jolie. In *ECOWS*, pages 13–22. IEEE Computer Society, 2007.
34. R. Neykova, L. Bocchi, and N. Yoshida. Timed runtime monitoring for multiparty conversations. In *BEAT*, volume 162 of *EPTCS*, pages 19–26, 2014.
35. H. R. Nielson, F. Nielson, and R. Vigo. A calculus for quality. In *FACS*, pages 188–204. Springer, 2013.
36. S. Pattem, B. Krishnamachari, and R. Govindan. The impact of spatial correlation on routing with compression in wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(4):24, 2008.
37. M. A. Perillo and W. B. Heinzelman. Wireless sensor network protocols. In A. Boukerche, editor, *Handbook of Algorithms for Wireless Networking and Mobile Computing*, pages 1–35. Chapman and Hall/CRC, 2005.
38. N. Yoshida, R. Hu, R. Neykova, and N. Ng. The scribble protocol language. In *TGC*, pages 22–41. Springer, 2013.

A Additional Definitions

A.1 Type System

Figure 9 presents the complete type system for GC_q.

Definition 3 (State satisfaction). *The entailment relation between a state σ and an environment Ψ, and the entailment relation between a state σ and a formula ψ are written $\sigma \models \Psi$ and $\sigma \models \psi$, respectively. They are defined as follows:*

$$
\begin{array}{ll}
\sigma \models \cdot & \iff \sigma \text{ is defined} \\
\sigma \models \psi, \Psi & \iff \sigma \models \psi \text{ and } \sigma \models \Psi \\
\sigma \models \mathtt{tt} & \iff \sigma \text{ is defined} \\
\sigma \models \mathtt{t} : k\,[A] \triangleright X & \iff (\mathtt{t}, k, X) \in \sigma \\
\sigma \models \psi_1 \otimes \psi_2 & \iff \sigma = \sigma', \sigma'' \mid \sigma' \models \psi_1 \ \wedge \ \sigma'' \models \psi_2
\end{array}
$$

Choreography Formation ($\Psi \vdash C$),

$$\frac{\Psi, \mathbf{init}(\mathtt{t}_r[\widetilde{A_r}]\{Y_r\}, \mathtt{t}_s[\widetilde{B_s}]\{Y_s\}, k) \vdash C \quad \{\widetilde{\mathtt{t}_s}, k\} \# (\mathbf{T}(\Psi) \cup \mathbf{K}(\Psi))}{\Psi \vdash \mathtt{t}_r[\widetilde{A_r}]\{Y_r\} \ \mathbf{start} \ \mathtt{t}_s[\widetilde{B_s}]\{Y_s\} : a(k); \ C} \lfloor \mathsf{Tinit} \rceil$$

$$\frac{\forall^{\geq 1} J.\ s.t. \left(\begin{array}{c} J \subseteq \widetilde{\mathtt{t}_r} \ \wedge \ q(J) \ \wedge \ \Psi = \psi_A, (\psi_j)_{j\in J}, \Psi' \ \wedge \ \phi = \mathtt{t}_A : k\,[A] \triangleright X_A \bigotimes_{j\in J} (\mathtt{t}_j : k\,[B_j] \triangleright X_j) \\ \wedge \ \phi' = \mathtt{t}_A : k\,[A] \triangleright Y_A \bigotimes_{j\in J} (\mathtt{t}_j : k\,[B_j] \triangleright Y_j) \ \wedge \ \psi_A, (\psi_j)_{j\in J}, \phi \multimap \phi' \vdash \phi' \end{array} \right) : \quad \mathtt{t}_A : k\,[A] \triangleright Y_A, (\mathtt{t}_j : k\,[B_j] \triangleright Y_j)_{j\in J}, \Psi' \vdash C \quad \vdash e@\mathtt{t}_A : \mathbf{opt.data} \quad \vdash x_i@\mathtt{t}_i : \mathbf{opt.data} \quad i\in\{1\ldots|\widetilde{\mathtt{t}_r}|\}}{\Psi \vdash \left(\mathtt{t}_A[A]\{X_A; Y_A\}.e \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_r[B_r]\{\widetilde{X_r; Y_r}\} : x_r) : k\right); \ C} \lfloor \mathsf{Tbcast} \rceil$$

$$\frac{\forall^{\geq 1} J.\ s.t. \left(\begin{array}{c} J \subseteq \widetilde{\mathtt{t}_r} \ \wedge \ q(J) \ \wedge \ \Psi = \psi_B, (\psi_j)_{j\in J}, \Psi' \ \wedge \ \phi = \mathtt{t}_B : k\,[B] \triangleright X_B \bigotimes_{j\in J} (\mathtt{t}_j : k\,[A_j] \triangleright X_j) \\ \wedge \ \phi' = \mathtt{t}_B : k\,[B] \triangleright Y_B \bigotimes_{j\in J} (\mathtt{t}_j : k\,[A_j] \triangleright Y_j) \ \wedge \ \psi_B, (\psi_j)_{j\in J}, \phi \multimap \phi' \vdash \phi' \end{array} \right) : \quad \mathtt{t}_B : k\,[B] \triangleright Y_B, (\mathtt{t}_j : k\,[A_j] \triangleright Y_j)_{j\in J}, \Psi' \vdash C \quad \vdash e_i@\mathtt{t}_i : \mathbf{opt.data} \quad \vdash x@\mathtt{t}_B : \mathbf{opt.data} \quad i\in\{1\ldots|\widetilde{\mathtt{t}_r}|\}}{\Psi \vdash \left(\&_{\mathbf{q}}(\mathtt{t}_r[A_r]\{\widetilde{X_r; Y_r}\}.e_r) \text{ -> } \mathtt{t}_B[B]\{X_B; Y_B\} : x : \langle k, op\rangle\right); \ C} \lfloor \mathsf{Tred} \rceil$$

$$\frac{\forall^{\geq 1} J.\ s.t. \left(\begin{array}{c} J \subseteq \widetilde{\mathtt{t}_r} \ \wedge \ q(J) \ \wedge \ \Psi = \psi_A, (\psi_j)_{j\in J}, \Psi' \ \wedge \ \phi = \mathtt{t}_A : k\,[A] \triangleright X_A \bigotimes_{j\in J} (\mathtt{t}_j : k\,[B_j] \triangleright X_j) \\ \wedge \ \phi' = \mathtt{t}_A : k\,[A] \triangleright Y_A \bigotimes_{j\in J} (\mathtt{t}_j : k\,[B_j] \triangleright Y_j) \ \wedge \ \psi_A, (\psi_j)_{j\in J}, \phi \multimap \phi' \vdash \phi' \end{array} \right) : \quad \mathtt{t}_A : k\,[A] \triangleright Y_A, (\mathtt{t}_j : k\,[B_j] \triangleright Y_j)_{j\in J}, \Psi' \vdash C}{\Psi \vdash \left(\mathtt{t}_A[A]\{X_A; Y_A\} \text{ -> } \&_{\mathbf{q}}(\mathtt{t}_r[\widetilde{B_r]\{X_r; Y_r\}}) : k[l_h]\right); \ C} \lfloor \mathsf{Tsel} \rceil$$

$$\frac{}{\Psi \vdash \mathbf{0}} \lfloor \mathsf{Tinact} \rceil \qquad \frac{\Psi \vdash C_1 \quad \Psi \vdash C_2}{\Psi \vdash e@\mathtt{t}?\, C_1 \ : \ C_2} \lfloor \mathsf{Tcond} \rceil \qquad \frac{\Psi = \psi \oplus \psi' \quad \psi \vdash C \quad \psi' \vdash C'}{\Psi \vdash C + C'} \lfloor \mathsf{Tsum} \rceil$$

Data Typing

$$\frac{}{\vdash t@p\colon \mathbf{data}} \lfloor \mathsf{TD1} \rceil \qquad \frac{}{\vdash v@p\colon \mathbf{data}} \lfloor \mathsf{TD2} \rceil$$

$$\frac{}{\vdash e@p\colon \mathbf{opt.data}} \lfloor \mathsf{TOD1} \rceil \qquad \frac{\vdash v\colon \mathbf{data}}{\vdash \mathsf{some}(v)@p\colon \mathbf{opt.data}} \lfloor \mathsf{TOD2} \rceil \qquad \frac{}{\vdash \mathsf{none}@p\colon \mathbf{opt.data}} \lfloor \mathsf{TOD3} \rceil$$

State Formation ($\sigma\colon \mathbf{state}$),

$$\frac{}{\emptyset\colon \mathbf{state}} \lfloor \mathsf{TS1} \rceil \qquad \frac{\sigma\colon \mathbf{state} \quad \sigma(\mathtt{t}[A], k) = \emptyset \quad X \in dom(\Sigma)}{\sigma, (\mathtt{t}[A], k, X)\colon \mathbf{state}} \lfloor \mathsf{TS2} \rceil$$

$$\frac{\sigma\colon \mathbf{state} \quad (\mathtt{t}[A], k, X) \in \sigma \quad Y \in dom(\Sigma)}{[\![X; Y]\!](\sigma(t, k))\colon \mathbf{state}} \lfloor \mathsf{TS3} \rceil \qquad \frac{\sigma\colon \mathbf{state} \quad \delta\colon \mathbf{state}}{\sigma \backslash \delta\colon \mathbf{state}} \lfloor \mathsf{TS4} \rceil$$

Formulae Formation ($\Psi\colon \mathbf{form}$),

$$\frac{}{\cdot\colon \mathbf{form}} \lfloor \mathsf{TF1} \rceil \qquad \frac{\psi\colon \mathbf{form} \quad \Psi\colon \mathbf{form}}{\psi, \Psi\colon \mathbf{form}} \lfloor \mathsf{TF2} \rceil \qquad \frac{}{\mathtt{tt}\colon \mathbf{form}} \lfloor \mathsf{TF3} \rceil \qquad \frac{}{\mathtt{t}\colon k\,[A] \triangleright X\colon \mathbf{form}} \lfloor \mathsf{TF4} \rceil$$

$$\frac{\psi\colon \mathbf{form} \quad \psi'\colon \mathbf{form} \quad \circ \in \{\otimes, \oplus\}}{\psi \circ \psi'\colon \mathbf{form}} \lfloor \mathsf{TF5} \rceil \qquad \frac{\psi\colon \mathbf{form} \quad \delta\colon \mathbf{state}}{\psi \backslash \delta\colon \mathbf{form}} \lfloor \mathsf{TF6} \rceil$$

Fig. 9: GC_q: Type checking - Complete rules

$$\begin{array}{ll} \sigma \models \psi_1 \oplus \psi_2 & \iff \sigma \models \psi_1 \ \mathit{or} \ \sigma \models \psi_2 \\ \sigma \models \psi \backslash \delta & \iff \exists \sigma' \ \mathit{s.t.}\ \sigma' \models \psi \ \wedge \ \sigma = \sigma' \backslash \delta \end{array}$$