Declarative Choreographies and Liveness*

Thomas T. Hildebrandt[1, orcid.org/0000-0002-7435-5563], Tijs Slaats[1, orcid.org/0000-0001-6244-6970], Hugo A. López[2,3, orcid.org/0000-0001-5162-7936], Søren Debois[2, orcid.org/0000-0002-4385-1409], and Marco Carbone[2, orcid.org/00000-0001-9479-2632]

[1] Software, Data, People & Society Section
Department of Computer Science
Copenhagen University, Denmark
hilde@di.ku.dk, slaats@di.ku.dk

[2] Department of Computer Science
IT University of Copenhagen, Denmark
hual@itu.dk, debois@itu.dk, maca@itu.dk

[3] DCR Solutions, Denmark

Abstract. We provide the first formal model for declarative choreographies, which is able to express general omega-regular liveness properties. We use the Dynamic Condition Response (DCR) graphs notation for both choreographies and end-points. We define end-point projection as a restriction of DCR graphs and derive the condition for end-point projectability from the causal relationships of the graph. We illustrate the results with a running example of a Buyer-Seller-Shipper protocol. All the examples are available for simulation in the online DCR workbench at http://dcr.tools/forte19.

Keywords: Choreographies, Liveness, Declarative Models

1 Introduction

Choreographies are an important tool for the development of highly distributed applications. Using an "Alice-talks-to-Bob" notation, they permit to abstract away details of a distributed implementation and focus on how the different components interact. This has been a fundamental driver for the adoption of choreographies in industry standards such as Message Sequence Charts (MSC) [22], UML Sequence Diagrams [32], WS-CDL, and BPMN Choreography Notation [31]. Moreover, choreography notations have been used in a range of application areas, including web-service development [6,7], synthesis of protocol behaviour [26], monitoring [3], parallel programming [27] and cyber-physical systems [28].

* **Acknowledgments:** Work supported by the Innovation Fund Denmark project *EcoKnow* (7050-00034A); the Danish Council for Independent Research project *Hybrid Business Process Management Technologies* (DFF-6111-00337), and the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement BehAPI No.778233.

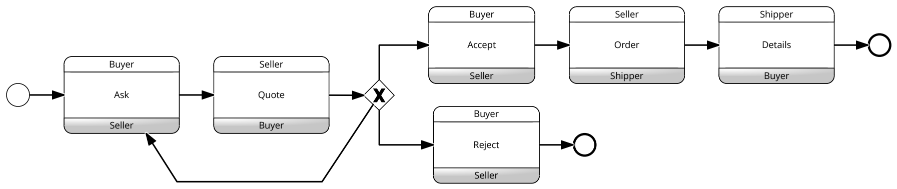

Fig. 1. BPMN Choreography for Buyer-Seller-Shipper example.

Paired with static analysis techniques (e.g., behavioural type systems), they are capable of deriving distributed (endpoint) implementations where endpoints generated from a choreography ascribe to all and only the behaviors defined by it. In practice, choreographies have "graduated" from the academic world, and several industrial programming languages implementing choreographies exist, e.g., [23,7,20].

A central aspect of choreography languages is the notion of *interaction.* An interaction is a first class citizen in any choreography language and, as a minimum, it collects information regarding the sender, the receivers, and the action used to synchronize participants. In Fig. 1, we show an exemplary BPMN choreography, based on a variant of the Buyer-Seller protocol [6]. The choreography involves three participants, a Buyer, a Seller and a Shipper. After asking the Seller for a quote and getting the reply, the Buyer may either Accept, Reject or Ask again. If the Buyer accepts, the Seller sends an Order to the Shipper, which subsequently sends the detailed confirmation directly to the Buyer.

The second aspect considered in the design of choreography languages is the ordering of interactions. Typically, choreography languages are seen as imperative programs as the BPMN choreography above, that describe *how* interactions should occur. Any other flow not explicitly written in the language is considered forbidden. It has been observed that imperative notations are often insufficiently flexible for modelling business processes[1,34]. An imperative notation focuses on describing a small number of ideal flows through a process. Adding more flows to represent edge cases and less common solutions to the model tends to increase its complexity significantly. While this approach works for processes where the ideal case is all we are interested in, this does not suffice for knowledge and case work: knowledge workers tend to deal with highly variant scenarios for which they need to determine unique solutions. For instance, in the choreography above, we may in practice really want the *liveness property*, that a Quote is eventually followed by a decision, but that the Sellers can provide new quotes or the Buyer can ask again, any finite number of times, before accepting or rejecting a quote. The present imperative choreography languages do not allow to specify such general liveness property.

In the present paper, we propose using the declarative Dynamic Condition Response (DCR) graph notation [29,35,15,9,30] as a formal declarative notation for both choreographies and end-point specifications, allowing the specification of both safety and general liveness properties. The DCR graphs notation has been developed for the formalisation and digitalisation of collaborative, adaptive case management processes. The notation is both supported by a range of formal techniques, and serves as the formal base for the industrial (`dcrgraphs.net`) design and simulation tool. During the recent years, the DCR graphs technology has been employed in major industrial case management systems used in the public sector in Denmark. DCR graphs have been extended to include both data [36], time[19] and sub-processes [9]. In the present paper we consider only the core notation, which is expressive enough to represent both regular and omega-regular languages [9] as well as so-called true concurrency [10]. This means that we provide the first choreography model supporting end-point projection and general liveness properties. Definition and simulation of DCR graphs is supported by the on-line DCR Workbench [12] available at `http://dcr.tools/forte19`. DCR diagrams in this paper were all produced using the workbench.

One of the important reasons for using choreography languages is their correctness-by-design guarantees. Message-passing distributed systems consist of communicating endpoints whose behaviours are defined in terms of input/output actions. So if the choreography is to be implemented by a message-passing distributed system, it is necessary to translate choreographies into code that can be executed by these endpoints. Such a translation is referred to as an endpoint projection. The endpoint projection, paired with the global properties of the choreography, warrants the safety of the distributed execution of the endpoints (e.g. deadlock-freedom). A catch of this approach is, however, that the choreography language often allows specifications that are not well-formed, meaning that it is not possible to realise the choreography as the composition of end-point processes. A key result for any choice of choreography language and end-point language is therefore to provide criteria for the choreography to be well-formed.

A core property is that of local causality. Intuitively, local causality means that if a participant initiates an interaction, it must not have direct dependencies (or causal relationships) to interactions in which this participant is not involved. The criteria for end-point projectability is, however, highly dependent of the chosen languages. In BPMN 2.0.2 it is formulated as a constraint on sequencing

> "The Initiator of a Choreography Activity MUST have been involved (as Initiator or Receiver) in the previous Choreography Activity."

as well as a number of more complex constraints on the use of so-called branching gateways in BPMN for choices. Proving the correctness of such criteria requires a formal semantics, which is not yet provided for BPMN Choreographies, but for similar notations [5]. In the present paper, we will build upon the formal semantics and theory of safe projections [18,19] for DCR graphs to provide end-point projections for DCR choreographies.

Summary of contributions: We provide a general end-point projection result for: 1) a declarative choreography model, 2) that can represent general omega-regular liveness properties [9], 3) supports a broad range of extensions such as dynamic process spanning and refinement [9], true-concurrency semantics [10] and time [19], and 4) is supported by both academic and industrial design and simulation tools.

2 Interactions and Dynamic Condition Response Graphs

In this section we first define the general concept of *interactions*, which are common to previous work on choreographies. We then recall the model of Dynamic Condition Response (DCR) graphs.

2.1 Interactions

Assume a fixed set of actions A, ranged over by a, b, c and a fixed set of roles R, ranged over by $r, r', r_1, r_2, \ldots$ (referred to as participants in [5]).

Definition 1. *An* interaction *is a triple* $(a, r \to R)$*, in which the action* $a \in \mathsf{A}$ *is* initiated *by the role* r *and* received *by the roles* $R \subset_{\mathsf{fin}} \mathsf{R}\backslash\{r\}$*, i.e a finite set of roles distinct from* r*. Define* $\mathsf{Initiator}((a, r \to R)) = r$*. We use the shorthand* $(a, r \to r')$ *for interactions between two participants* $(a, r \to \{r'\})$*. We denote by* IA *the set of all interactions.*

We proceed to define projections of interactions to actions for end-point processes. End-point processes describe the view of the process from a single participant r synchronising with the other participants via messages on channels: For each interaction $(a, r \to R)$ in the choreography, there will be channels $(a, r \to r')$ from r to r' for each $r' \in R$. To ease the definition of projections and avoid introducing new notation, we describe actions for an end-point also as interactions. That is, for the end-point process at role r, we use the interaction $(a, r \to R')$ to represent the action $!(a, r \to R')$ for sending a message on the channels $(a, r \to r')$ for all participants $r' \in R'$. The interaction $(a, r' \to r)$, represents the action $?(a, r' \to r)$ for receiving a message on the channel $(a, r' \to r)$. We apologize to the reader for the inconvenience this reuse of notation may cause.

Definition 2. *For an interaction* $\alpha = (a, r' \to R')$*, define the* end-point projection of α at r *by:*

$$\alpha_{|r} = \begin{cases} (a, r' \to r) & \text{when } r \in R' \\ (a, r' \to R') & \text{when } r' = r \\ \tau & \text{otherwise} \end{cases} \tag{1}$$

We extend end-point projections to sets and sequences of interactions by point-wise projection and removing τ actions, and finally to sets of sequences of interactions in the obvious way.

Definition 3. *A* choreographic language *is a triple* (C, A, R) *where* $A \subseteq \mathsf{A}$*,* $R \subseteq_{fin} \mathsf{R}$*, and* $C \subseteq \mathsf{IA}^\infty = \mathsf{IA}^* \cup \mathsf{IA}^\omega$*. That is, a set* C *of finite and infinite sequences of interactions for a given set of actions* A *and roles* R*.*

When A and R are obvious from the context, we shall take C as defining a choreographic language.

Definition 4. *The* end-point projection *of a choreographic language* (C, A, R) *is the family of languages* $(C_{|r})_{r \in R}$*.*

2.2 DCR Graphs

In this section we recall Dynamic Condition Response (DCR) graphs [14,15,29,13,12]. This paper follows the set-based formulation of [15,29,13].

As formally defined below, a DCR graph consists of a directed graph and a marking. The nodes of the graph are labelled events and the edges are relations of five kinds: conditions ($\rightarrow\!\bullet$), responses ($\bullet\!\!\rightarrow$), inclusions ($\rightarrow\!+$), exclusions ($\rightarrow\!\%$) and milestones ($\rightarrow\!\diamond$).

Definition 5. *A* DCR graph *is a tuple* $(E, M, L, \ell, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$*, where*

- E *is a set of events*
- $M \subseteq E \times E \times E$ *is a* marking
- L *is a set of* labels
- $\ell : E \rightarrow L$ *is a* labelling function
- $\phi \subseteq E \times E$ *for* $\phi \in \{\rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%\}$ *are* relations *between events.*

A DCR graph defines a process whose executions are finite and infinite sequences of (labelled) events. Note that an event may be executed several times. The three sets of events in the marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ defines the state of the DCR graph process, and are referred to as the **ex***ecuted events* (Ex), the *pending* **re***sponse*[4] *events* (Re) and the **in**cluded events (In). The relations define effects of the execution of events and constrain the executions of the process defined by the DCR graph as defined formally below. Briefly:

- An inclusion (respectively exclusion) relation $e \rightarrow\!+ e'$ (respectively $e \rightarrow\!\% e'$) means that if e is executed, then e' is included (respectively excluded).
- A condition relation $e \rightarrow\!\bullet\ e'$ means that e is a condition for e', i.e. if e is included, then e must have been executed for e' to be enabled for execution.
- A response relation $e \bullet\!\!\rightarrow e'$ means that whenever e is executed, e' becomes a pending response. During a process execution, a pending event must eventually be executed (which makes it no longer pending, unless it has a response relation to itself) or be excluded. We refer to e' as a response to e.
- A milestone relation $e \rightarrow\!\diamond\ e'$ means that if e is included it must not be pending for e' to be enabled for execution. We refer to e as a milestone for e'. Milestones are typically used in cyclic behaviour, when some earlier executed event e may be required to be executed again, i.e. it becomes pending, before the process can proceed executing event e'.

For DCR graph G with events E and marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ and event $e \in E$ we write $(\rightarrow\!\bullet e)$ for the set $\{e' \in E \mid e' \rightarrow\!\bullet\ e\}$, write $(e\bullet\!\!\rightarrow)$ for the set $\{e' \in E \mid e \bullet\!\!\rightarrow e'\}$ and similarly for $(e\rightarrow\!+)$, $(e\rightarrow\!\%)$ and $(\rightarrow\!\diamond e)$. We can now define when the events of a DCR graph are *enabled*.

Definition 6 (Enabled events). *Let* $G = (E, M, L, \ell, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$ *be a DCR graph, with marking* $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. An event* $e \in \mathsf{E}$ *is* enabled*, written* $e \in \mathsf{enabled}(G)$*, iff (a)* $e \in \mathsf{In}$ *and (b)* $\mathsf{In} \cap (\rightarrow\!\bullet e) \subseteq \mathsf{Ex}$ *and (c)* $(\mathsf{Re} \cap \mathsf{In}) \cap (\rightarrow\!\diamond e) = \emptyset$*.*

[4] We often simply say ”pending” instead of ”pending response”.

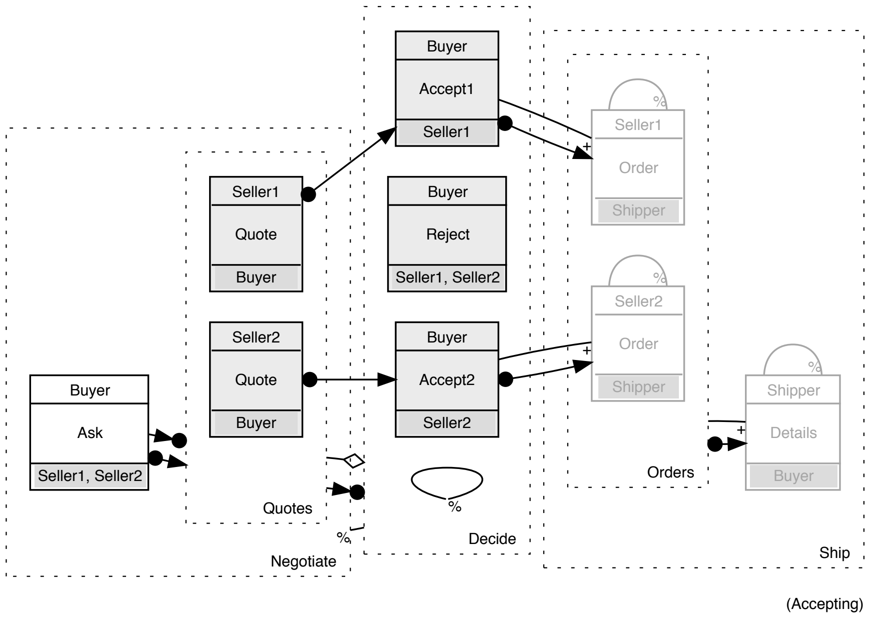

Fig. 2. Example DCR choreography

That is, enabled events (a) are included, (b) their included conditions have already been executed, and (c) have no pending included milestones.

Example 7. We give an example of a DCR graph in Fig. 2 as visualised by the online-tool `dcr.tools/forte19`. Events are indicated by boxes with solid borders and collections of events are shown with dashed boxes. Relations are shown as arrows between the boxes. As formalised in [17], such collections are referred to as ”nestings” and are just a visual shorthand, understanding arrows to (from) nestings to represent arrows to (from) *every* event inside the nesting.

Traditionally, the labels of DCR graphs only consist of an action and possibly a set of roles that may perform the action. In the present paper, however, the labels of the DCR graphs are interactions rather than just actions. Instead of labelling the boxes representing the events simply with the label, e.g., $(\mathsf{Ask}, \mathsf{Buyer} \rightarrow \{\mathsf{Seller1}, \mathsf{Seller1}\})$, we have split the label in three fields in the visualisation similarly to the notation for BPMN choreographies: The initiator (Buyer) is written in the field at the top of the box; the action (Ask) is written in the middle field, and the receiver(s) $(\mathsf{Seller1}, \mathsf{Seller2})$ in the bottom field with a grey background. When no confusion is possible, we refer to events by the action shown in middle field, speaking of, e.g., “the event Ask” rather than the more precise “the event labelled Buyer, Ask, Seller1,Seller2.”

The marking of the graph and whether events are enabled or not is indicated visually: If the background of all fields is grey for a box, the event is included, but not enabled. For instance, the Ask event is enabled, but the two Quote events are not enabled (because the Ask event is a condition for the events and not yet executed). A box which is made opaque/dimmed out, such as the two Order events and the Details event, represents an event which is not included. When explaining Fig. 3 we describe the visualisation of executed and pending events.

The response relation ($\bullet\!\!\rightarrow$) from the event Ask to the nesting box around the Quote events means that the Quote events become pending when Ask is executed. This expresses the liveness constraint, that if the buyer asks, a quote must eventually be given by both sellers. The milestone relation ($\rightarrow\!\diamond$) from the box around the Quote events to the nesting box labelled Decide means that the events (Accept1, Accept2 and Reject) inside the Decide box cannot be executed if any of the Quote events are pending, not even if Quote happened in the past. This expresses the safety condition, that the buyer can not accept or reject if one of the sellers has not responded after the last time the buyer asked for a quote. The inclusion relation from e.g. Accept1 to Order for Seller 1 means that order event will be included if the buyer accepts the quote from Seller 1. The circular exclusion arrow in the Decide box means that any event inside the box is related by an exclude relation to any event inside the box, i.e. they are mutually and self-exclusive. That is, whenever one of the events inside the box happens, all three events inside the box are excluded. Moreover, due to the exclude relation from the Decide box to the Negotiate box, also the Ask and Quote events are excluded when a decision is made.

Below we formalise how the marking changes when an enabled event e is *executed*: (a) the event e is added to the set of executed events, (b) e is removed from the set of pending response events, and the responses to e are added to the set of pending response events, (c) the events excluded by e are removed from the set of included events, and the events included by e are added to the set of included events.

Definition 8 (Execution). *Let* $G = (E, M, L, \ell, \rightarrow\!\bullet, \bullet\!\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$ *be a DCR graph, with marking* $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$*. When* $e \in \mathsf{enabled}(G)$*, the result of executing* e*, written* $\mathsf{execute}(G, e)$ *is a new DCR graph* G' *with the same events, labels, labelling function and relations, but a new marking* $M' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$*, where (a)* $\mathsf{Ex}' = \mathsf{Ex} \cup \{e\}$ *(b)* $\mathsf{Re}' = (\mathsf{Re} \backslash \{e\}) \cup (e\bullet\!\!\rightarrow)$*, and (c)* $\mathsf{In}' = (\mathsf{In} \backslash (e\rightarrow\!\%)) \cup (e\rightarrow\!+)$

Example 9. In the graph in Fig. 2, we may execute the event Ask. Following the relations in the graph, this will make the two Quote events enabled and pending: they were previously not enabled due to their condition relation from Ask and once Ask becomes "executed" in the marking, that condition is fulfilled, and Quote becomes enabled. Altogether, executing Ask yields the graph shown in Fig. 3. Red text and an exclamation mark after the action in the middle field represents an event which is pending, as is the case for the two Quote events. A box with a check mark after the label represents an event which is executed, as can be seen for the event Ask. Note that Ask may be executed again immediately, leaving the process in the same state, but the two Quote events must eventually be executed (without any intermediate Ask) in order for the run to be accepting.

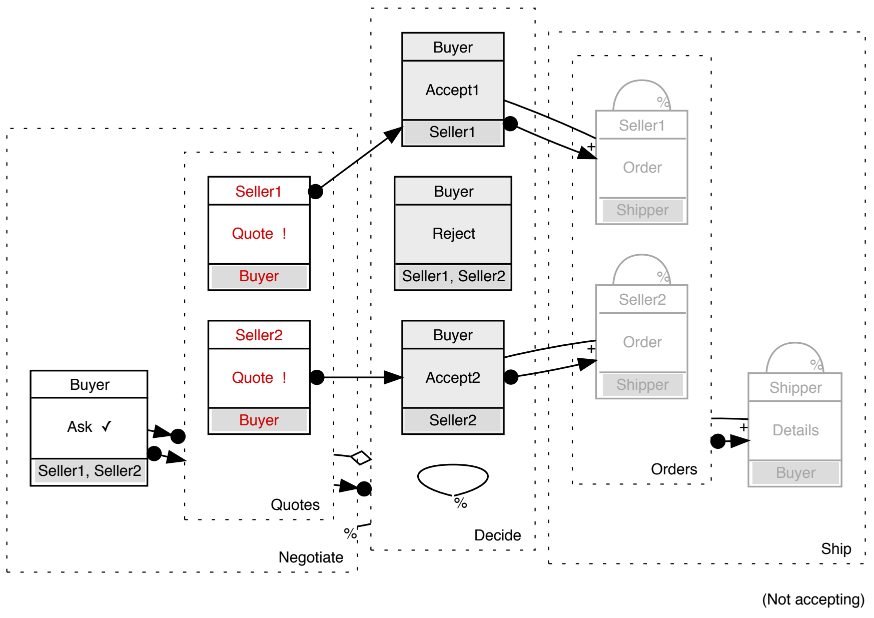

Fig. 3. DCR choreography after execution of Ask.

From the definition of execution we can define a transition semantics for DCR graphs using labelled event transition system with responses.

Definition 10 (Transition semantics). *Let* $G = (E, M, L, \ell, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$ *be a DCR graph. The* Labelled Event Transition System with Responses *(LETSR) for* G *is defined as* $\mathcal{T}(G) = (\mathcal{G}, G, E, L, \ell, \rightarrow, \rho)$*, where the DCR graph* G *is the initial state,* E *is the set of events,* L *is the set of labels,* ℓ *is the labelling function,* $\rightarrow \subseteq \mathcal{G} \times E \times \mathcal{G}$ *is the transition relation, defined by* $(G, e, G') \in \rightarrow$ *iff* $e \in \mathsf{enabled}(G)$ *and* $G' = \mathsf{execute}(G, e)$*, and* $\mathcal{G} = \{G' \mid G \rightarrow^* G'\}$*, the set of states, is the graphs reachable from the initial graph* G *by execution of events, and finally* ρ *is the response function defined on DCR graphs by* $\rho(G') = \mathsf{Re} \cap \mathsf{In}$*, if* $(\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ *is the marking of* G'*.*

We say that two LETSR T and T' are isomorphic, written $T \equiv T'$, if there is an isomorphism between the sets of states preserving and respecting transitions and the response function.[5]

We define the language of a DCR graph as all finite and infinite sequences of such executions, where we demand that all pending responses are either eventually executed or excluded.

[5] Isomorphism could be defined more generally by also having an isomorphism on the set of events, but the given definition is sufficient for the present paper.

Definition 11 (Language of a DCR graph). *Let $G = (E, M, L, \ell, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$ be a DCR graph. A* run *of G is a finite or infinite sequence of events $e_0, e_1, \ldots$ such that $e_i \in \mathsf{enabled}(G_i)$, $\mathsf{execute}(G_i, e_i) = G_{i+1}$, and $G_0 = G$. We call a run* accepting *iff for each G_i with marking $M_i = (\mathsf{Ex}_i, \mathsf{Re}_i, \mathsf{In}_i)$ and $e \in \mathsf{Re}_i \cap \mathsf{In}_i$ there exists a $j \geq i$ such that $e_j = e$ or $e \notin \mathsf{Re}_j \cap \mathsf{In}_j$.*

The language *$\mathsf{lang}(G) \subseteq L^\infty$ of G is the set of finite and infinite sequences of labels $l_0 l_1 \cdots$ such that there is an accepting run $e_0, e_1, \ldots$ where $\ell(e_i) = l_i$.*

It has been proven in [9] that DCR graphs can express exactly the languages that are the union of a regular and an ω-regular language. This means that one can express regular safety and liveness properties in DCR graphs.

Since the definition of accepting runs only depends on the included pending responses in the markings of the graphs and the events being executed during a run, it is easy to see that if two DCR graphs have isomorphic transition systems with responses then they also have the same languages.

Proposition 12. *Let G and G' be DCR graphs. If $\mathcal{T}(G) \equiv \mathcal{T}(G')$ then $\mathsf{lang}(G) = \mathsf{lang}(G')$.*

3 DCR Choreographies

Below we first account for how DCR Choreographies and DCR End-points can be defined using DCR graphs. We then derive the criteria for end-point projectability and provide the operational correspondence between an end-point projectable DCR graph and the synchronous composition of its end-points.

Definition 13. *Let $A \subseteq \mathsf{A}$ and $R \subset_{fin} \mathsf{R}$ be sets of roles and actions, respectively. A triple (G, A, R) is then a* DCR choreography *when G is a deadlock-free DCR graph such that the labels $L \subseteq \mathsf{IA}$ of G are interactions with actions in A and participants in R. For a role $r \in R$, a tuple (G, A, R, r) is a* DCR End-point *when the labels L of G are interactions either of the form $(a, r \rightarrow R')$ or $(a, r' \rightarrow r)$ with $a \in A$.*

Example 14. The DCR graph in Fig. 2 is the DCR graph G of a DCR choreography (G, A, R) where the actions A and roles R are given by:

$$A = \{\mathsf{Ask}, \mathsf{Quote}, \mathsf{Accept1}, \mathsf{Accept2}, \mathsf{Reject}, \mathsf{Order}, \mathsf{Details}\}$$
$$R = \{\mathsf{Buyer}, \mathsf{Seller}, \mathsf{Shipper}\}$$

Note that by virtue of being a general declarative notation, one may specify DCR graphs with deadlocks, e.g. by having a cycle of condition relations. It is easy to prove, that if $(\rightarrow\!\bullet \cup \rightarrow\!\diamond)$, i.e. the union of the condition and milestone relations, is acyclic, then the DCR graph is free of deadlocks. Moreover, such graphs can express all languages expressed by general DCR graphs, and in particular the complex behaviour in our running example.

We now turn to the key question for any choreography language: How do we project a global choreography description onto the intended behaviour of individual participants? And in particular, is this operation always possible, or are some global descriptions in fact not realisable by individual end-points?

Projections and distributed execution have been studied for DCR graphs in previous work [18,16,19], but in a rather different setting where events are only labeled by actions and initiating roles, not recieving roles. For this reason, it is safe in [18,16,19] to leave out an event in the projection to an end-point, if the execution of this event does not impact the state or enabledness of any event initiated by the participant responsible for that end-point. An example of such an event in our running example is the Details event for the Buyer end-point.

In the present paper we have explicit receivers and need to preserve all receiving events for a participant. Consequently, we can not directly use the notion of projection given in [18,16,19]. However, we may yet *build* on these projections to obtain one useful for DCR choreographies. The core idea in those papers was to project a graph G with events E to a network of local graphs for any division (not necessarily disjoint) of the events $\delta_1 \cup \delta_2 \cup \ldots \cup \delta_n = E$, then define synchronous composition of such networks of DCR graphs. Intuitively, shared events, i.e. events occurring in more than one graph, are executed synchronously in the network, representing communication. The projection then ensured that execution of the network formed by the local graphs would have a transition system isomorphic to that of the global graph, and thus in particular exhibit the same language as the global graph.

In the following we reconcile the notion of projection from [18,16,19] and then subsequently define end-point projections.

First, we characterise when the execution of an event may change the marking or enabledness of another. To this end, we define the notion of *direct dependency.*

Definition 15. *Let $G = (E, M, L, \ell, \to\!\bullet, \bullet\!\to, \to\!\diamond, \to\!+, \to\!\%)$ be a DCR graph and let $e, e' \in E$ be events of E. Then there is a* direct dependency $e' \preceq e$ *from e' to e iff either of the following conditions are true*

1. $e' = e$,
2. $e'(\to\!\bullet \cup \bullet\!\to \cup \to\!+ \cup \to\!\% \cup \to\!\diamond)e$,
3. $\exists e''.\ e'(\to\!+ \cup \to\!\%)e''(\to\!\bullet \cup \to\!\diamond)e$,
4. $\exists e''.\ e' \bullet\!\to e'' \to\!\diamond e$.

That is, $e' \preceq e$ iff either (1) they are the same, (2) there is a relation from e' to e, (3) e' includes or excludes an event which is itself a condition or milestone for e, or (4) e' has a response to a milestone for e.

The following proposition states that an event e *must* be directly dependent on any event e' whose execution may change the marking or enabledness of e.

Proposition 16. *Let G be a DCR graph with marking $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$. Suppose $e' \in \mathsf{enabled}(G)$, and let $G' = \mathsf{execute}(G, e')$ and $M' = (\mathsf{Ex}', \mathsf{Re}', \mathsf{In}')$ be the marking of G'. If either of the following hold, then $e' \preceq e$.*

1. *$e \in \mathsf{enabled}(G) \not\Leftrightarrow e \in \mathsf{enabled}(G')$,*
2. *$e \in \mathsf{Ex} \not\Leftrightarrow e \in \mathsf{Ex}'$,*
3. *$e \in \mathsf{Re} \not\Leftrightarrow e \in \mathsf{Re}'$,*
4. *$e \in \mathsf{In} \not\Leftrightarrow e \in \mathsf{In}'$.*

Proof. (Sketch) For lack of space we just show why condition 2 above implies $e' \preceq e$, the other conditions follow from a similar inspection of the definitions. First note that the set Ex of executed events always grows, i.e. once executed an event can never become not executed. So we only need to consider the case $e \notin \mathsf{Ex}$ and $e \in \mathsf{Ex}'$. From the Def. 8 it is clear that the only event that can be included in the set Ex during execution is the event being executed, so it follows that $e = e'$ and thus $e' \preceq e$.

We note that this implication is not a bi-implication, e.g., in the DCR graph comprising just the two events e, f and the single relation $e \to\bullet f$ in a marking where e is already executed, we clearly have $e \preceq f$ (by Definition 15(2) because there is a relation from e to f), yet executing e in fact cause *no* changes to marking or enabledness of f.

Intuitively, we will obtain the end-point projection for a participant r by keeping (a) events labelled with interactions involving r, *as well as* (b) the direct dependencies of the events for which r is the initiator. We then interpret the interactions as end-point actions as described in Sec. 2.1. In order for the interactions to make sense as actions for the end-point process at r, the role r must be involved in its direct dependencies. We formalise this as follows.

Definition 17. *Let (G, A, R) be a DCR choreography and ℓ the labelling function of G; and let $r \in R$ be a role. This choreography is* end-point projectable *for r iff for all e, if $\mathsf{Initiator}(e) = r$ and $e' \preceq e$, then $\ell(e')_{|r} \neq \tau$,*

Example 18. Referring again to the example DCR choreography in Fig. 2, we find that this choreography is in fact *not* end-point projectable for the participants Seller1 and Seller2. We see in Fig. 2 that the Accept1 event causes $(\mathsf{Quote}, \mathsf{Seller2} \to \mathsf{Buyer})$ to be excluded, having Seller2 as initiator, but *not* participating in Accept1. To be precise, because of the exclusion we have $\mathsf{Accept1} \preceq (\mathsf{Quote}, \mathsf{Seller2} \to \mathsf{Buyer})$ and $\ell(\mathsf{Accept2}) = (\mathsf{Accept2}, \mathsf{Buyer} \to \mathsf{Seller2})$ yet we have $\ell(\mathsf{Accept1}) = (\mathsf{Accept1}, \mathsf{Buyer} \to \mathsf{Seller1})$, so $\ell(\mathsf{Accept1})_{|\mathsf{Seller2}} = \tau$.

We fix this by redefining the choreography such that Seller2 is included in the Accept1 interaction, that is, so that Seller2 is notified that he lost the contract; and vice versa including Seller1 in the Accept2 interaction. We show the projectable process in Fig. 4. This choreography *is* end-point projectable.

We proceed by defining the end-point projection. We start by recalling the definition of projection in [18,16,19], adapted to keep all labels.

Definition 19 (Adapted DCR δ-Projection cf. [18,16]). *Given a DCR graph*

$$G = (E, M, L, \ell, \to\bullet, \bullet\to, \to\diamond, \to+, \to\%)$$

and a set of events $\delta \subseteq E$, define the projection of G to the events δ as the graph $G|_\delta = (E|_\delta, M|_\delta, L|_\delta, \ell|_\delta, \to\bullet|_\delta, \bullet\to|_\delta, \to\diamond|_\delta, \to+|_\delta, \to\%|_\delta)$ given by:

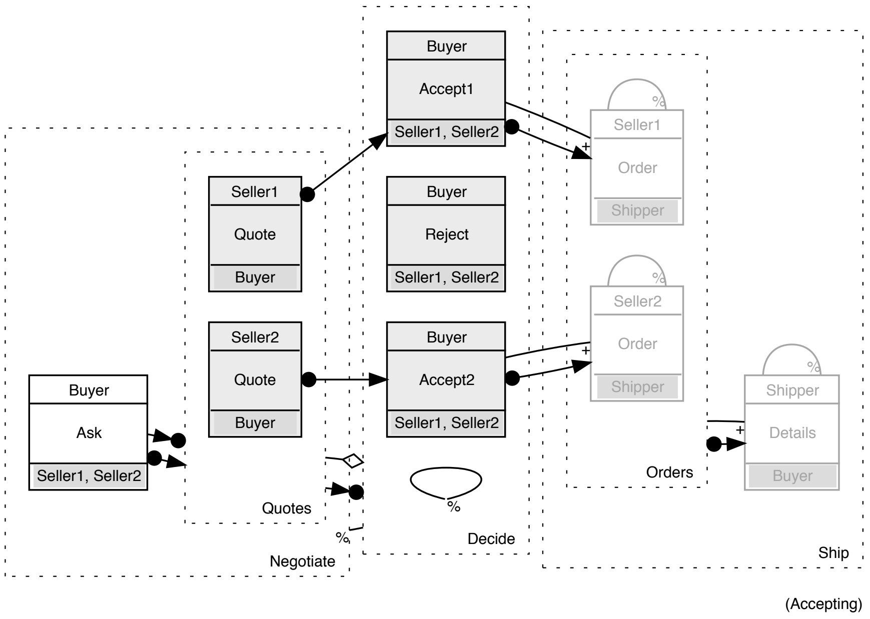

Fig. 4. End-point projectable DCR choreography

1. $E|_\delta = \{e \in E \mid \exists e' \in \delta.\ e \preceq e'\}$,
2. $M|_\delta = (\mathsf{Ex}|_\delta, \mathsf{Re}|_\delta, \mathsf{In}|_\delta)$ *where:*
   (a) $\mathsf{Ex}|_\delta = \mathsf{Ex} \cap E|_\delta$
   (b) $\mathsf{Re}|_\delta = \mathsf{Re} \cap E|_\delta$
   (c) $\mathsf{In}|_\delta = \big(\mathsf{In} \cap ((\to\!\bullet\, \delta) \cup\ (\to\!\diamond\, \delta) \cup\ \delta)\big) \cup \big(E|_\delta \setminus ((\to\!\bullet \delta) \cup (\to\!\diamond \delta) \cup \delta)\big)$.
3. $\ell|_\delta(e) = \ell(e)$,
4. $L|_\delta = \mathsf{img}(\ell)$
5. $\to\!\bullet|_\delta = \to\!\bullet \cap \big((\to\!\bullet\ \delta) \times \delta\big)$
6. $\to\!\diamond|_\delta = \to\!\diamond \cap \big((\to\!\diamond\ \delta) \times \delta\big)$
7. $\bullet\!\!\to|_\delta = \bullet\!\!\to \cap \big(((\bullet\!\!\to\to\!\diamond\ \delta) \times (\to\!\diamond\ \delta)) \cup ((\bullet\!\!\to\ \delta) \times \delta)\big)$
8. $\to\!+|_\delta = \to\!+ \cap(((\to\!+\!\to\!\bullet\ \delta) \times (\to\!\bullet\ \delta)) \cup ((\to\!+\!\to\!\diamond\ \delta) \times (\to\!\diamond\ \delta)) \cup ((\to\!+\ \delta) \times \delta))$

9. $\to\!\%|_\delta = \to\!\% \cap(((\to\!\%\!\to\!\bullet\ \delta) \times (\to\!\bullet\ \delta)) \cup ((\to\!\%\!\to\!\diamond\ \delta) \times (\to\!\diamond\ \delta)) \cup ((\to\!\%\ \delta) \times \delta))$

The complexity in these rules arises mostly from the necessity of including events that may affect milestones or conditions for the events in δ. We see this particularly in the right-most half of 2(c), in the second clause in 7–9, and in the fourth clause in 8–9.

We now define the end-point projection for a DCR choreography with respect to a role r. The projection comes in two steps: first we compute the $\delta-$projection, taking δ to be the set of events for which r is the initiator. Second, we simply add all events where r is a receiver. The latter step does not really change the behaviour in terms of sequences of actions, but it ensures that all receiving roles will be present for an interaction, even when it has no effect on other events in the end-point. As also described in the beginning of the section, this was not essential for the previous work, since receivers were not explicit.

Definition 20 (DCR end-point projection). *Let (G, A, R) be a DCR choreography with events E and labelling function ℓ. For any $r \in R$, define*

$$\delta = \{e \in E \mid \mathsf{Initiator}(e) = r\} ,$$

and let $G_{|\delta} = (E_{|\delta}, M, L, \ell_{|\delta}, \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%)$ be the δ-projection of G for r. Suppose $M = (\mathsf{Ex}, \mathsf{Re}, \mathsf{In})$ and define

$$\begin{aligned} E' &= \{e \in E \mid \exists a\, r'.\ \ell(e)_{|r} = (a, r', r)\} \\ M' &= (\emptyset, \emptyset, E' \backslash (E \backslash \mathsf{In})) \\ \ell'(e) &= \begin{cases} \ell(e)_{|r} & \textit{if } e \in E_{|\delta} \cup E' \\ \mathsf{undefined} & \textit{otherwise} \end{cases} \end{aligned}$$

The end-point projection of (G, A, R) for r *is then defined as the DCR end-point $(G_{|r}, A_{|r}, R, r)$ where*

$$G_{|r} = (E_{|\delta} \cup E', M \cup M', \mathsf{img}(\ell'), \ell', \rightarrow\!\bullet, \bullet\!\rightarrow, \rightarrow\!\diamond, \rightarrow\!+, \rightarrow\!\%).$$

Example 21. The result of end-point projecting the corrected choreography in Fig. 4 can be seen in Fig. 5, Fig. 6 and Fig. 7.

Lemma 22. *Let (G, A, R) be a DCR choreography, let $r \in R$ be a role of R, and let $(G_{|r}, A_{|r}, R, r)$ be the projection of that choreography to r. If (G, A, R) is projectable for r, then every label in $G_{|r}$ is an interaction which has r as a participant.*

Proof. The set of events of $G_{|r}$ consists of the events $E_{|\delta}$ of the δ-projection for $\delta = \{e \in E_0 \mid \mathsf{Initiator}(e) = r\}$ and the events $E' = \{e \in E \mid \exists a\, r'.\ \ell(e)_{|r} = (a, r', r)\}$. Clearly, the events in E' by definition all have the role r among the receivers and thus as participant. According to Def. 19, we have $E|_\delta = \{e \in E \mid \exists e' \in \delta.\ e \preceq e'\}$. Now, since $\mathsf{Initiator}(e') = r$ for all $e' \in \delta$ it follows from the definition of end-point projectability in Def. 17 that $\ell(e)_{|r} \neq \tau$ when $e \preceq e'$ and thus r is also a participant in the interaction for all $e \in E|_\delta$.

As shown below, it follows easily, that if an event is shared between two end-points, it has the same initiator.

Lemma 23. *Let $C = (G, A, R)$ be an end-point projectable DCR choreography, $r, r' \in R$ be roles of R, and $(G_{|r}, A_{|r}, R, r)$ and $(G_{|r'}, A_{|r'}, R, r')$ be the projections of C to r and r'. If $e \in E \cap E'$, where E and E' are the events of $G_{|r}$ and $G_{|r'}$ respectively, then $\mathsf{Initiator}(\ell(e)) = \mathsf{Initiator}(\ell(e'))$.*

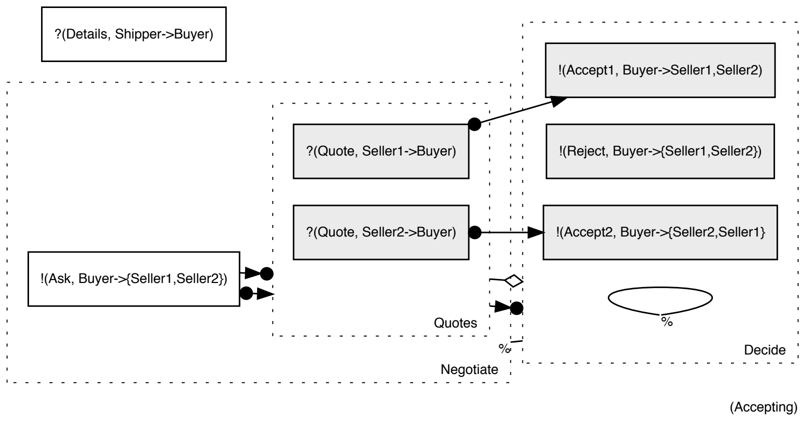
?(Details, Shipper->Buyer)
!(Accept1, Buyer->Seller1,Seller2)
?(Quote, Seller1->Buyer)
!(Reject, Buyer->{Seller1,Seller2})
?(Quote, Seller2->Buyer)
!(Accept2, Buyer->{Seller2,Seller1}
!(Ask, Buyer->{Seller1,Seller2})
%
Quotes
%
Decide
Negotiate
(Accepting)

Fig. 5. End-point projection of Fig. 4 for Buyer.

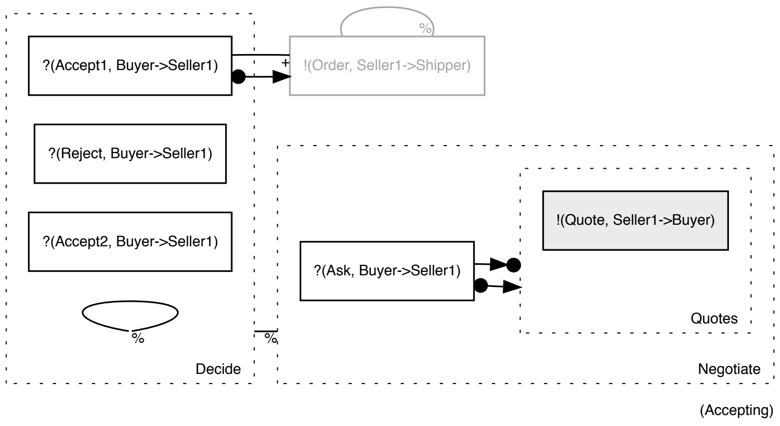
%
?(Accept1, Buyer->Seller1)
+
!(Order, Seller1->Shipper)
?(Reject, Buyer->Seller1)
!(Quote, Seller1->Buyer)
?(Accept2, Buyer->Seller1)
?(Ask, Buyer->Seller1)
%
%
Quotes
Decide
Negotiate
(Accepting)

Fig. 6. End-point projection of Fig. 4 for Seller1.

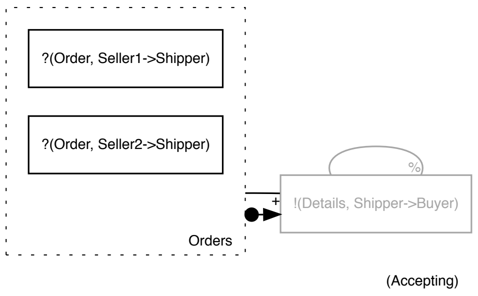
?(Order, Seller1->Shipper)
?(Order, Seller2->Shipper)
%
+
!(Details, Shipper->Buyer)
Orders
(Accepting)

Fig. 7. End-point projection of Fig. 4 for Shipper.

Proof. *According to Def. 20, the label of an event e in an end-point projection for role r is given by the restriction $\ell(e)_{|r}$. According to Def. 2 the initiator role is preserved by the restriction or the label is τ. However, by Def. 17 the label cannot be τ. Thus, being end-point projections of the same end-point projectable choreography C the event in the two end-point projections will have the same initiator role.*

We now define the synchronous composition of a finite set of DCR end-points, for which the labels of shared events agree on the Initiator role. Intuitively, an event e is enabled in the synchronous composition, if it is enabled in all of the end-points in which it occurs. The execution of an event is then defined simply by executing the event in all of the components it occurs. Finally, the label is the interaction obtained by taking the union of receivers.

Definition 24 (Synchronous composition of DCR end-points). *For $R = \{r_1, r_2, \ldots, r_n\}$ and DCR end-points $P_i = (G_i, A_i, R, r_i)$ for $i \in \{1,..,n\}$ we write the synchronous parallel composition as $P = \Pi_{i\in\{1,..,n\}} P_i$. Define*

- *$E = \bigcup_i \in \{1,..,n\} E_i$, where E_i is the events of G_i.*
- *$e \in \mathsf{enabled}(P)$ iff $e \in E_i$ implies $e \in \mathsf{enabled}(G_i)$ for all $i \in \{1,..,n\}$.*
- *$\mathsf{execute}(P, e) = \Pi_{i\in\{1,..,n\}} P'_i$, if $e \in \mathsf{enabled}(P)$ and $P_i = (G'_i, A_i, R, r_i)$ and $G'_i = \mathsf{execute}(G_i, e)$, if $e \in E_i$ and $P'_i = P_i$ otherwise.*
- *$\ell_P(e) = (a, r \to R')$ if $e \in E_i$ implies $\ell_i(e) = (a, r \to R'_i)$ and $R' = \bigcup_i \in I$, where $I = \{i \in \{1, \ldots, n\} \mid e \in E_i\}$.*
- *$\rho_P(P') = \bigcup_{i\in\{1,\ldots,n\}} \mathsf{Re}_i \cap \mathsf{In}_i$ if $P' = \Pi_{i\in\{1,..,n\}} P'_i$, $P'_i = (G'_i, A_i, R, r_i)$ and the marking of G'_i is $(\mathsf{Ex}_i, \mathsf{Re}_i, \mathsf{In}_i)$.*

We now define the LETSR for P by $\mathcal{T}(P) = (\mathcal{P}, P, E, L, \ell_P, \to_P, \rho_P)$, where $(P', e, P'') \in \to_P$ if $e \in \mathsf{enabled}(P')$ and $P'' = \mathsf{execute}(P', e)$, and $\mathcal{P} = \{P' \mid P \to^* P'\}$.

The following theorem establishes the key property, that the synchronous composition of the end-points yields a transition system with responses isomorphic to the transition system for the choreography, and thus is in particular deadlock free.

Theorem 25. *Let $C = (G, A, R)$ be an end-point projectable DCR choreography, $R = \{r_1, \ldots, r_n\}$ and $P_i = (G_i, A_i, R, r_i)$ for $i \in \{1, \ldots, n\}$ the DCR end-points resulting from end-point projection of C. Then $\mathcal{T}(C) \equiv \mathcal{T}(\Pi_{i\in\{1,..,n\}} P_i)$ and thus $\Pi_{i\in\{1,..,n\}} P_i$ is deadlock free.*

Proof. *(Sketch) The proof follows the same approach as the proof of Thm.5.1 in [19] where a bisimulation is constructed between the original graph (in this case the choreography) and the network of synchronous parallel composition of projections. The reason why the same approach can be used is that the main difference between the present work and the work in [19] is that we have included also receiving events in the end-points that have no effect on the synchronous product, such as the* Details *event in Fig. 5.*

We note that the isomorphism by Prop. 12 implies that the language of the choreography is the same as the language of the composition of the end-points.

4 Conclusion and Related Work

Based on the formal process notation of DCR graphs, we have provided the first declarative model for choreographies able to describe general liveness properties. We identified the local causality criteria for end-point projectability and defined end-point projections, using previous work on distributions of DCR graphs. We showed that the synchronous product of the end-point projections had the same behaviour as the original choreography. As future work we intend to extend the results to declarative timed choreographies, benefiting from projections already being defined for timed DCR graphs in [19].

Related Work. Properties for guaranteeing projectability are proposed in various settings and depend on the chosen choreography language. The results in [6] require three main properties: connectedness, well-threadedness, and coherence. While well-threadedness and coherence concern the behaviour of replicated servers, connectedness is the same as the projectability criterium of BPMN 2.0.2, which we also adopt in the present paper. The connectedness property occurs also in other works on choreography, e.g., [24,8]. In the theory of multiparty session types [21] and in Chor [7], such property is omitted at a price of a more flexible interpretation of sequencing.

To the authors' knowledge, this is the first work considering general liveness properties at the choreography-level. Other works in the literature have studied liveness for multiparty interactions from session types and contract development: Padovani et al. [33] propose a type system for session types to control liveness properties. However, the model considered is roleless since types describe interactions but without specifying which roles implement them. The work in [11] extends binary session types to specify response properties, that is applied to a variant of a collaborative BPMN process language to verify whether liveness for dead-lock free processes can be achieved. A recent paper by Lange et al. [25] investigates a bounded liveness property for GO programs, where protocols are specified as global types. Such property resembles a progress property and is not as general as our liveness. For instance, requiring that after a Quote we have eventually an Accept or a Reject cannot be expressed as bounded liveness. *Honesty* is a variant of liveness used in contract-oriented programming [2]. In short, an endpoint is honest if it abides the sequence of actions stipulated in its contract. Honesty will fail if the contract promises the execution of an action and the endpoint does not execute it. Contracts in this sense correspond to DCR responses. We differ from [2] in the sense that we do not require a session-type to verify liveness. It is specified in the model as a behavioural constraint only in the places that is required.

Our previous work in [4], presented a proof system for choreographies where properties such as liveness and connectedness can be expressed in terms of modal (may/must) operators. Apart from the difference on the languages explored (the global calculus in [4] and DCR in the present work), we differ from being able to express one-to-many communications. For DCR graphs, projections were studied in a different context in [18,16,19], where all participants were implicit receivers of actions and projections thus always defined.

References

1. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer (2006)
2. Bartoletti, M., Scalas, A., Tuosto, E., Zunino, R.: Honesty by typing. Logical Methods in Computer Science 12(4) (2016)
3. Bocchi, L., Chen, T., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. In: FMOODS/FORTE. LNCS, vol. 7892, pp. 50–65. Springer (2013)
4. Carbone, M., Grohmann, D., Hildebrandt, T.T., López, H.A.: A logic for choreographies. In: PLACES. EPTCS, vol. 69, pp. 29–43 (2010)
5. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centred programming for web services. In: ESOP. vol. 4421 (2007)
6. Carbone, M., Honda, K., Yoshida, N.: Structured communication-centered programming for web services. ACM Trans. Program. Lang. Syst. 34(2), 8:1–8:78 (2012)
7. Carbone, M., Montesi, F.: Deadlock-freedom-by-design: Multiparty asynchronous global programming. In: Symposium on Principles of Programming Languages. pp. 263–274. POPL '13, ACM, New York, NY, USA (2013)
8. Cruz-Filipe, L., Montesi, F., Peressotti, M.: Communications in choreographies, revisited. In: ACM Symposium on Applied Computing. pp. 1248–1255. ACM (2018)
9. Debois, S., Hildebrandt, T., Slaats, T.: Safety, liveness and run-time refinement for modular process-aware information systems with dynamic sub processes. In: FM 2015. pp. 143–160. No. 9109 in LNCS, Springer (2015)
10. Debois, S., Hildebrandt, T., Slaats, T.: Concurrency and asynchrony in declarative workflows. In: Business Process Management (BPM). LNCS, vol. 9253. Springer, Cham (2016)
11. Debois, S., Hildebrandt, T.T., Slaats, T., Yoshida, N.: Type-checking liveness for collaborative processes with bounded and unbounded recursion. Logical Methods in Computer Science 12(1) (2016)
12. Debois, S., Hildebrandt, T.: The DCR Workbench: Declarative Choreographies for Collaborative Processes. In: Gay, S., Ravara, A. (eds.) Behavioural Types: from Theory to Tools, pp. 99–124. River Publishers (Jun 2017)
13. Debois, S., Hildebrandt, T.T., Slaats, T.: Concurrency and Asynchrony in Declarative Workflows. In: Business Process Management (BPM). LNCS, vol. 9253, pp. 72–89. Springer (2015)
14. Debois, S., Hildebrandt, T.T., Slaats, T.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Informatica 55(6), 489–520 (Sep 2018)
15. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: PLACES. EPTCS, vol. 69, pp. 59–73 (2010)
16. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Declarative modelling and safe distribution of healthcare workflows. In: Foundations of Health Informatics Engineering and Systems - First International Symposium, FHIES. pp. 39–56 (2011)
17. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Nested Dynamic Condition Response Graphs. In: Fundamentals of Software Engineering, FSEN. LNCS, vol. 7141, pp. 343–350. Springer (Apr 2011)
18. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T.: Safe distribution of declarative processes. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM. LNCS, vol. 7041, pp. 237–252. Springer (2011)

19. Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. Journal of Logic and Algebraic Programming 82(5-7), 164–185 (2013)
20. Honda, K., Mukhamedov, A., Brown, G., Chen, T., Yoshida, N.: Scribbling interactions with a formal foundation. In: ICDCIT. LNCS, vol. 6536, pp. 55–75. Springer (2011)
21. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. J. ACM 63(1), 9:1–9:67 (2016)
22. ITU recommendation z.120 : Message Sequence Chart (MSC) (August 2011), `https://www.itu.int/rec/T-REC-Z.120-201102-I/en`
23. Kouzapas, D., Dardha, O., Perera, R., Gay, S.J.: Typechecking protocols with mungo and stmungo: A session type toolchain for java. Sci. Comput. Program. 155, 52–75 (2018)
24. Lanese, I., Guidi, C., Montesi, F., Zavattaro, G.: Bridging the gap between interaction- and process-oriented choreographies. In: Intl. Conf. on Software Engineering and Formal Methods, SEFM. pp. 323–332 (2008)
25. Lange, J., Ng, N., Toninho, B., Yoshida, N.: Fencing off go: liveness and safety for channel-based programming. In: POPL. pp. 748–761. ACM (2017)
26. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: POPL. pp. 221–232. ACM (2015)
27. López, H.A., Marques, E.R.B., Martins, F., Ng, N., Santos, C., Vasconcelos, V.T., Yoshida, N.: Protocol-based verification of message-passing parallel programs. In: OOPSLA. pp. 280–298. ACM (2015)
28. López, H.A., Nielson, F., Nielson, H.R.: Enforcing availability in failure-aware communicating systems. In: FORTE. LNCS, vol. 9688, pp. 195–211. Springer (2016)
29. Mukkamala, R.R.: A Formal Model For Declarative Workflows: Dynamic Condition Response Graphs. Ph.D. thesis, IT University of Copenhagen (June 2012)
30. Mukkamala, R.R., Hildebrandt, T., Slaats, T.: Towards trustworthy adaptive case management with dynamic condition response graphs. In: EDOC. pp. 127–136. IEEE (2013)
31. Object Management Group BPMN Technical Committee: Business Process Model and Notation, version 2.0.2 (2014), `http://www.omg.org/spec/BPMN/2.0.2/PDF`
32. Object Management Group UML Technical Committee: Unified Modeling Language, version 2.5.1 (2017), `http://www.omg.org/spec/UML/2.5.1/`
33. Padovani, L., Vasconcelos, V.T., Vieira, H.T.: Typing liveness in multiparty communicating systems. In: COORDINATION. LNCS, vol. 8459, pp. 147–162. Springer (2014)
34. Reijers, H.A., Slaats, T., Stahl, C.: Declarative modeling–an academic dream or the future for bpm? In: Daniel, F., Wang, J., Weber, B. (eds.) Business Process Management, LNCS, vol. 8094, pp. 307–322. Springer Berlin Heidelberg (2013)
35. Slaats, T.: Flexible Process Notations for Cross-organizational Case Management Systems. Ph.D. thesis, IT University of Copenhagen (January 2015)
36. Strømsted, R., López, H.A., Debois, S., Marquard, M.: Dynamic evaluation forms using declarative modeling. In: Proceedings of the Dissertation Award and Demonstration, Industrial Track at BPM 2018 (2018), `CEUR-WS.org`