Trace-based Verification of Rule-based Service Choreographies

Sotiris K. Moschoyiannis  
Department of Computer Science  
University of Surrey  
Guildford, GU2 7XH, UK  
s.moschoyiannis@surrey.ac.uk

Leandros Maglaras  
Department of Computing Technology  
De Montfort University  
Leicester, LE1 9BH, UK  
leandros.maglaras@dmu.ac.uk

Nurulhuda A Manaf  
Department of Mathematics  
National Defence University of Malaysia  
57000 Kuala Lumpur, Malaysia  
nurulhuda@upnm.edu.my

Abstract—The service choreography approach has been proposed for describing the global ordering constraints on the observable message exchanges between participant services in service oriented architectures. Recent work advocates the use of structured natural language, in the form of Semantics of Business Vocabulary and Rules (SBVR), for specifying and validating choreographies. This paper addresses the verification of choreographies - whether the local behaviours of the individual participants conform to the global protocol prescribed by the choreography. We describe how declarative specifications of service choreographies can be verified using a trace-based model, namely an adaptation of Shields’ vector languages. We also use the so-called blackboard rules, which draw upon the Bach coordination language, as a middleware that adds reactivity to this declarative setting. Vector languages are to trace languages what matrices are to linear transformations; they afford a more concrete representation which has advantages when it comes to computation or manipulation.

Keywords—SBVR; service interactions; verification; vector languages; concurrency; global behaviour; complex systems

I. INTRODUCTION

The coordination of distributed systems comprising autonomous participants (e.g., stand alone services) is inherently challenging. Sustained efforts by the web services community have culminated in the service choreography approach [1] which is concerned with describing the conversation across different participating entities (global perspective) as well as the service orchestration approach [2] which describes the interaction scenario from an individual service’s viewpoint. Choreography in particular, is intended to capture the coordination of the participant services in terms of observable message exchanges between them.

The difficulty in choreography design lies with the need to mediate between conformance and flexibility (e.g., see [3]). On the one hand, all participant services must respect the agreed global constraints while on the other hand each participant should be able to execute its part of the choreography as freely as possible, preserving its local autonomy and the replaceability of services. There needs to be provision for checking whether specific (service) requests can be fulfilled in a given service choreography, e.g., see [4] for checking choreographies (expressed declaratively) using the standard constraint solver Alloy Analyzer. Formal work for service choreography [3], [5]–[7] is typically concerned with the following verification task: given a choreography specification, does a set of local behaviours of participant services generate those and only those behaviours in the specification, or equivalently, does the behaviour of each participant service conform to the specification?

The necessity of conceptual modelling of interactions has resulted in different modelling language proposals such as Business Process Model and Notation (BPMN) [8], Web Services Business Process Execution Language (WS-BPEL) [2], Web Services Choreography Description Language (WS-CDL) [1]. More recently, choreography diagrams were included in BPMN 2.0 [8], and have been used for modelling RESTful services interactions in [9], [10] uses RESTful services in the cloud to increase reliability, while [11] studies fault tolerance. OMG’s UML 2 design models [12] are widely used for specifying service choreographies (e.g., [13], [3]). Proposals so far in this direction tend to require specific training and hence do not target the end-user directly.

The OMG standard Semantics of Business Vocabulary and Rules (SBVR) [14] is gaining ground as a basis for service oriented systems specification [4], [15]–[17]. The idea is to capture specifications in natural language and represent them in formal logic so they can also be machine processed. In this way, end-users are able to validate the spec directly, which can then be parsed and executed by a machine. SBVR was proposed for specifying service choreographies in [17].

In this paper, we present an approach to the verification of service choreographies which have been specified declaratively using an SBVR model and Blackboard (BB) rules [18]. By design SBVR does not include time. [17] has proposed a notion of ‘precedence’ whilst staying within the OMG specification [14]. The BB rules bring in a stricter notion of precedence, more akin to immediate causality.

The semantics for verifying whether a set of individual behaviours conform to a global protocol is typically based on traces [3], [6] or bisimulation [5], [7]. Our approach uses a trace-based model, namely an adaptation of Shields’ vector languages [19] in that the synchronisation constraint is
removed (no shared actions). The vector languages notation captures the messages on each participant (e.g., calls on service interfaces, REST operations over HTTP), and at each point during execution. It is a model of true concurrency which is appropriate for modelling service choreographies where it is not reasonable to assume the presence of a central orchestrator that can serialise concurrent interactions.

The contribution of this paper is in the verification of declarative choreography specifications, using vector languages. First, a UML 2 design model is used to capture the message exchange, which can be translated into vector languages following the semantics given in [20]. We then build an SBVR model of the same choreography, augmented by BB rules. We show how this model generates a vector language, essentially tuples of sequences, one sequence for each participant. The behaviours in the generated language are then validated against those in the language obtained directly from the choreography specification.

The remainder of this paper is structured as follows. Section II provides the preliminaries necessary for this approach. Section III presents the specification of a choreography in terms of both UML 2 sequence diagrams and the rule-based model. Section IV describes how this rule-based model gives rise to a vector language capturing both local and global behaviours. Section V discusses related work while Section VI concludes the paper.

II. PRELIMINARIES

Here, we briefly outline key aspects of the OMG standard SBVR [14], the blackboard rules approach to coordination [21] and the trace-based model of vector languages [19], [22], which provide the background to the approach.

A. Semantics of Business Vocabulary and Rules (SBVR)

SBVR is a meta-model with models natively expressed as logical formulations. Its most common serialization is SBVR Structured English (SBVR - SE) [14], which provides a standardized representation to formalize the syntax of natural language representation. Terms (a noun concept, e.g., branch), Fact Types (e.g., rental car is owned by branch), and rules (e.g. It is obligatory that each rental car is owned by at least one branch) are combined into models. The colouring in the font is prescribed in the OMG SBVR specification [14]. In line with the Business Rules Approach [23], SBVR models follow the doctrine: ”[...] Terms express business concepts; facts make assertions about these concepts; rules constrain and support these facts.”

The SBVR rule given earlier is a representation of higher-level facts that use the deontic constraint obligatory on the constraint defined by the rule. The quantification, each and at least one effect restriction of the rental car belonging. Furthermore, is owned by is the designation for the Fact Type which makes an assertion on the corresponding Terms (here, rental car and branch).

Thus, the combination of the deontic constraint, quantifier, Terms and Fact Types will yield a constructive rule. This type of rule can be used by domain specialists (e.g., business analysts) in defining the business model or activity to be performed with a service choreography. The resulting specification, expressed in the structured english of an SBVR model, can be validated by the same domain specialists as opposed to the implementation specialists. A notion of precedence between Fact Types has been proposed in [17], which is useful in the verification of a service choreography, in terms of the orderings of observable messages at each local platform and the global constraints.

B. Blackboard (BB) rules

The so-called Blackboard (BB) [24] approach has been proposed as a medium for communicating information (read, write) between distributed resources (services) in a system. Our interest in this approach stems from the fact that certain variants of it can express that one action immediately precedes another, i.e., immediately causality. Of the four types of BB architecture originally proposed in [25], we focus on Blackboard Server as it resembles the set up of stand-alone interacting services.

There are two primitives that can appear in a rule: \(in(a, u, t_1)\) denotes that the information \(u\) will become available on blackboard \(a\) at time \(t_1\) while \(nin(b, v, t_2)\) denotes the information \(v\) will no longer be available on blackboard \(b\) at time \(t_2\). In this paper, we will be consider forward reading in BB rules, which means reading from the left-hand side (LHS) of a rule to its right-hand side (RHS). The LHS becomes active when the context defined is verified. The RHS rule gets triggered after the LHS has been activated. For example:

\[in(a, [\text{location, Brussels}], t_1) \rightarrow in(a, [\text{list, museum}], t_2)\]

which says that ”if the information that the location is Brussels is on BB \(a\) at time point \(t_1\), then the information list of museums will be on BB \(a\) at time point \(t_2\)”. In effect, LHS and RHS are verifiable statements on actions while \(\rightarrow\) denotes immediate causality. There are two cases that need to be considered. If the action(s) in RHS need to occur immediately after the LHS is verified, then time need not be specified in RHS - execution of its action is triggered once the LHS has been verified. Otherwise, time is specified in the RHS.

Two notions of time are considered for BB rules [18]: i) discrete time refers to specific time points, i.e., \(t_k(1 \leq k \leq q)\), and ii) continuous time, which is either absolute or relative. Absolute time specifies that for each \(in\) primitive, at least one instance of the information \(u_i\), for \((1 \leq i \leq n)\), must be available on the blackboard within the interval time \([t_{i,1}, t_{i,2}]\). Similarly, for the \(nin\) primitive. In contrast, relative time specifies that for each \(in\) primitive, at least one instance of the information \(u_i\), for \((1 \leq i \leq n)\), must
be available within the duration $d_i$ time frame appearing on the respective blackboard. Similarly, for \( \text{nin} \).

C. Basics of Vector Languages

Apart from the declarative approaches there is a long strand of work on formalisms that describe the behaviour of systems by listing all sequences of actions names (traces). Typically, a process is associated with a set \( M \) of sequences, and the set of all behaviours of this process is given by the set of all prefixes of elements of \( M^* \). Proposals tend to differ on account of the notation used and the treatment of concurrency [26].

For the purposes of formally describing service choreography it seems appropriate to focus on vector languages [19], [22], [27] since this semantics is defined in terms of tuples of sequences, one for each process (participant service). The behaviour of a set of participants indexed by some set \( I \) is represented by a set of sequences indexed by the same set \( I \), which record the actions (message send / receive) experienced by the individual process (participant) during execution. Next, we give a brief account of this trace-based model. The interested reader is referred to [19], [22], [27] for full details.

In modelling service choreography we are interested in the coordination of a set of sequential process (participant services), so a choreography is associated with an indexed family \((C)_{i \in I}\) of such processes. Each process is associated with a set \( A_i \) of actions (or messages). A choreography is thus associated with an indexed cover in the following sense.

**Definition 1:** Let \( A \) and \( I \) be sets. An \( I \)-indexed cover on \( A \) is a pair \((A, \alpha)\) where \( \alpha : I \rightarrow \wp(A) \) is such that \( \bigcup_{i \in I} \alpha(i) = A \). We define \( V_\alpha \) to be the set of all functions \( \varphi : I \rightarrow A^* \) such that for all \( i \in I, \varphi(i) \in \alpha(i) \).

We refer to members of \( V_\alpha \) as vectors. The vector \( \varphi \) satisfying \( \varphi(i) = \Lambda \), each \( i \), is written as \( \Lambda_i \) where \( \Lambda \) denotes the empty sequence. When \( I = \{1, \ldots, n\} \), a vector in \( V_\alpha \) may be represented by the tuple

\[
\varphi(1), \ldots, \varphi(n) \in \alpha(1)^* \times \cdots \times \alpha(n)^*
\]

Such vectors may be treated in much the same way as conventional sequences. In particular, the elements of \( V_\alpha \) can be concatenated and compared.

**Definition 2:** If \((A, \alpha)\) is an \( I \)-indexed cover, then we define a binary operation \( \cdot \) and a relation \( \leq \) on \( V_\alpha \) as follows.
\[
1) \quad (\varphi, \psi) = \varphi \cdot \psi \quad \text{if} \quad \varphi(i) = \psi(i)
\]
\[
2) \quad \varphi \leq \psi \iff \forall i, \varphi(i) \leq \psi(i)
\]
where \( \cdot \) denotes concatenation of sequences and \( \leq \) is the prefix ordering on sequences \( (x \leq y \iff \exists z : xz = y) \). By arguing coordinate-wise, it is straightforward to show that \( V_\alpha \) is (i) a monoid with binary operation concatenation and identity element \( \Lambda \), and (ii) a partially ordered set with respect to prefix ordering with bottom element \( \Lambda \). Note that \( V_\alpha \) itself is not a free monoid. It is essentially the direct product of the free monoids \( \alpha(i)^* \).

**Definition 3:** If \((A, \alpha)\) is an \( I \)-indexed cover, \( a \in A \), then we define \( A \in V_\alpha \) by

\[
A(i) = \begin{cases} 
\alpha(i), & a \in \alpha(i) \\
\Lambda, & a \notin \alpha(i)
\end{cases}
\]

We denote the set of all such functions by \( A_\alpha \) and denote the submonoid of \( V_\alpha \) generated by \( A_\alpha \) under concatenation by \( A_\alpha^* \). The elements of \( A_\alpha \) are referred to as action vectors while the elements of \( A_\alpha^* \) as behaviour vectors. For the purposes of verifying choreographies in this paper, we will be interested in the submonoid \( A_\alpha^* \) generated by the design model for the choreography, e.g., BPMN 2.0 choreography diagram [8] or UML2 sequence diagram [12] (cf Fig. 2).

Again, \( A_\alpha^* \) is not a free monoid. In fact, certain action vectors may commute, and this is central to describing true concurrency within the order structure of a vector language \( A_\alpha^* \). This may be seen more clearly perhaps in the associated Hasse diagrams of Fig. 1. In (i) actions \( a \) and \( b \) are ordered \( b \) only occurs after \( a \), in (ii) they are concurrent \((a \text{ \ and \ } b \text{\ take \ place \ concurrently}) \), and in (iii) \( a \) and \( b \) are mutually exclusive \((\text{there is a choice between } a \text{ \ and } b)\). We return to this discussion in the next section (Section III-A).

III. DECLARATIVE SPECIFICATION OF SERVICE CHOREOGRAPHIES

We start by modelling a service choreography using standard UML design models. We then show how it can be specified declaratively using SBVR. The business rules in the SBVR model are then transformed to BB rules. This paves the way for verification (Section IV). We use a well known case study to illustrate our approach.

A. Acme Travel: a case study

We model the hypothetical travel scenario in the Acme Travel case study from [28], also studied in [29] in view of declarative specification of service choreographies and the multi-party conversation involved in arranging travel. In brief, a customer sends a reservation request which may include one or more of airline, accommodation (hotel or apartment), transport (bus or train or taxi), and tour reservations. Once Acme Travel (AT) receives the itinerary request it sends reservation requests to the different providers and awaits for responses. Once all (un)successful reservations are in, AT sends a notification to the customer.
Note that most constraints infer an explicit ordering between the specified activities and associated interactions (messages), whether these are service calls (RPC-style) or invocations on RESTful APIs. These interactions are captured in the UML2 sequence diagram [12] shown in Figure 2. This provides a global definition of the common ordering constraints within the multi-party conversation (see conversational part of W3C’s WS-CDL [1]). The work in [20] describes a formal translation of UML 2 sequence diagrams for restricting to the appropriate submonoid $\Lambda_{\alpha}^*$ generated by $A_{\alpha}$, incidentally giving a true concurrency semantics to the par interaction fragment found in UML 2 [12]. This means we can distinguish between concurrency and nondeterminism in the specification. The formal construction of the translation is given in [20].

The very start of the lifelines is mapped onto the empty vector $\Lambda = (\Lambda, \Lambda, \Lambda, \Lambda, \Lambda)$. The interactions described in the sequence diagram are translated into a set of behaviour vectors (Definition 1), as shown in Fig. 2. Due to space limitations we omit further details and refer to [20] for a comprehensive treatment of the formal translation. It can be readily applied to obtain a formal description of the choreography in terms of the trace-based model of vector languages. The key outcome of the translation is that the orderings between actions on different participants manifest themselves in the order structure of the set of vectors obtained from the choreography described in the UML2 sequence diagram.

It can be seen in Fig. 2 that messages $a$ and $h$ are sent in no particular order by $AT$ and $AR$, respectively. The fact these messages are unordered is denoted by the par interaction fragment. Meanwhile, the responses from $AR$ and $HR$, $r_1$ and $r_2$, respectively, are received by $AT$ in either order. The requirements specification does not imply some ordering on $r_1$ and $r_2$. However, these messages are received on the same participant service. Hence, they can be handled according to the policies of the local platform (e.g., FIFO queues as in Mealy services [3]). To remain faithful to the requirements specification we have allowed for $r_1$, $r_2$ to be received in either order, so as not to impose one ordering over another.

It might also be worth pointing out the specification prescribes that $a$ and $h$ are concurrent. These messages are received at different participant services, hence it is not reasonable to expect them to be serialised; at least not without compromising the loose coupling between the participant services. The subtle difference in the handling of the two situations described above is evident in the order structure of the corresponding vector language shown in Fig. 2 (right).

The receiving of $b$ and $a$ is modelled as independent actions that occur consecutively, after some behaviour, and their occurrence leads to the common behaviour captured by the vector $(b, a, h, \Lambda)$ (sitting at the bottom of the lozenge in the Hasse diagram on Fig. 2). The concurrent actions $a$ and $b$ are followed by a choice between ‘$r_1$ followed by $r_2$’ and ‘$r_2$ followed by $r_1$’, i.e., the possible serialisations of the unordered events $r_1$, and $r_2$, which is reminiscent of an interleaving semantics.

B. Using an SBVR model for service choreography

In what follows we outline certain aspects of the SBVR model built for this case study

In order to model service chains and choreographies we need to capture participant services (participants) and the messages exchanged between them (events). These will be Terms in the Vocabulary of the SBVR model. For example, Term: participant, Term: customer, Term: reservation request. Recent work in [17] advocates the use of SBVR in specifying service choreographies, focusing on the ordering of the interactions between participant services. The informal constraints in the business requirements III-A can be represented in an SBVR model in the form of Terms and Fact Types, which are then used to construct the rules. Further details can be found in [17].

Next, Fact Types and the SBVR construct Term verb Term can be used to capture the sending and the receiving of a message. For example, the export message of customer in the Fact Type: customer sends reservation request and the import message of AT in the Fact Type: AT receives reservation request capture the interaction between them.

As mentioned previously SBVR Structured English (SBVR-SE) [14] has a logical formulation which draws from first order logic (when ignoring the modalities). The logical connectives for exclusive disjunction (XOR), conjunction (AND), and inclusive disjunction (OR) are used on participants, events (Terms) as well as Fact Types in forming Business Rules in an SBVR model. For example, the general form when unordered events are concerned is the following: Rule: It is obligatory that [...] exactly one participant sends exactly one event 1 and exactly one event 2. The conjunction is used to express that both events take place, in no particular order.

1See https://tinyurl.com/y9t8x4cy for the full blown SBVR model.
We note that the Fact Type ‘acme-travel receives booking request is of customer’ is the synonymous form, as defined in the SBVR standard [14] of the Fact Type ‘customer sends booking request to acme-travel’. This says they convey the same meaning - both describe that the Acme Travel service receives a booking request from a customer service. This also applies to other rules that dictate the sending / receiving of messages in the choreography. We wish to keep a lean set of rules that constrain a choreography, hence we do not need rules that do not add meaning.

It is worth noting that WS-CDL [1] focuses on the receiver of the information, hence the WS-CDL Interaction Activity descriptions refer to the activity performed when the information is received. Our interest is in the observable events in service choreography and thus actions can be understood as invocations on the RESTful interfaces of the participant services, or, more generally, invocations on coordinator components as shown in the scenario of Fig. 2. This means it is appropriate to restrict only to consider the receiving of a message as an observable event. Hence, we keep the ‘receives’ Fact Types.

The work in [17] also introduced a notion of precedence between Fact Types appearing in a Rule. This captures the ordering in terms of dependency (causality) between certain messages (causality) in a choreography. For instance, it is obligatory that the customer receives a charge for the booking request from the acme-travel. The customer receives the successful notification of the booking request from the acme-travel specifies that the customer receives a charge for their booking before they receive a successful notification for that booking.

Furthermore, the usage of the verb concept receives as mentioned before enables to express the recipient among the service participants, e.g., in the above rule, the Acme Travel service receives the booking request, and Airline Reservation and Hotel Reservation services receive an airline and a hotel request, respectively, from the Acme Travel service.

C. BB rules for specifying immediate successors.

In the context of choreography verification, and with reference to Section II, the interest is in specifying the immediately succeeding action, hence time on the RHS of BB rules need not be specified. We apply both primitives in and nin to specify whether the information will be available or not on the blackboard at the specified time. In addition, we consider discrete time points which express the time of each primitive as a number depicting the units of time relative to the current time point. This is because the discrete time may define the information of an ordered list of activities / actions. A BB rule in the specification can only be triggered once. Thus, the absolute time has also been considered to express at least one tuple of information presented on the blackboard. This will be discussed in more detail in Section IV.

Table I shows the global constraints for the Acme Travel case study that have been transformed from the SBVR rules. For example, \( [msg, b, t_1] \) denotes \( b \) as the message of a booking request that must be present on the blackboard BB at time \( t_1 \). Each primitive in a BB rule represents the execution of an event, where \( b \) represents an event of booking request, \( a \) is an event of airline booking request, \( h \) is an event of hotel booking request, \( r_1 \) is a response of airline booking request event, \( r_2 \) is a response of hotel booking request event, \( c \) is an event of charge for Customer. \( n \) is a notification event, and \( s \) is a successful booking event.

<table>
<thead>
<tr>
<th>Blackboard BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>in(BB, [msg, b], ( t_1 )) ( \rightarrow ) in(BB, [msg, a]), in(BB, [msg, h])</td>
</tr>
<tr>
<td>in(BB, [msg, a], ( t_1 )) ( \rightarrow ) in(BB, [msg, h], ( t_2 ))</td>
</tr>
<tr>
<td>nin(BB, [msg, h], ( t_2 ))</td>
</tr>
<tr>
<td>in(BB, [msg, r_1], ( t_2 )) ( \rightarrow ) in(BB, [msg, r_2])</td>
</tr>
<tr>
<td>in(BB, [msg, r_2], ( t_3 )) ( \rightarrow ) in(BB, [msg, c])</td>
</tr>
<tr>
<td>in(BB, [msg, c], ( t_4 )) ( \rightarrow ) in(BB, [msg, s])</td>
</tr>
</tbody>
</table>

Table I: Global constraints expressed as BB rules

For example, the BB rule

\[ \text{in}(BB, [msg, b], t_1) \rightarrow \text{in}(BB, [msg, a]), \text{in}(BB, [msg, h]) \]

can be split into the LHS which denotes the Fact Type ‘acme-travel receives booking request is of customer’, the ‘\( \rightarrow \)’ which represents a more strict ‘precedes’ to show the immediate successor (event), and the RHS which denotes the Fact Type ‘airline-reservation receives airline booking request of customer and hotel-reservation receives hotel booking request of customer from acme-travel’.

Each \( \text{in} \) primitive in BB rules plays the role of a Fact Type in SBVR, but while it says what action immediately precedes another, it does not say which participant service is involved in the interaction. Hence, by the combination of the BB and SBVR rule, the BB rule can be read (forward reading) as: ‘if the information that the Acme Travel service receives a booking request from the customer is on the blackboard BB at \( t_1 \), then the information that the Airline Reservation receives an airline booking request and the Hotel Reservation service receives a hotel booking request from Acme Travel will be available on the blackboard’.

IV. MAPPING ONTO VECTOR LANGUAGES

In this section, we describe how a vector language can be generated from the rule-based SBVR (and BB) specification of a service choreography. This vector language is then validated against that obtained from the sequence diagram (recall Fig. 2).

Step 1: Identifying participant services and their alphabets. The Fact Types (FTs) in the SBVR model of a service choreography are used to identify the participants and the set of action names (messages) associated with each. As explained in Section III we only consider the FTs
that use the verb concept receives. More specifically, the pattern participant receives event is used in this step. So from the FT acme-travel receives booking request of customer we may deduce that acme-travel is a participant while booking request is a message received by Acme Travel, hence an action we want to capture in the trace-based model. Following this through the SBVR model, results in identifying four participant services each with its own set of messages, as shown in Table II.

<table>
<thead>
<tr>
<th>Participant Service</th>
<th>Set of Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>{booking request (denoted by b), response for airline booking request (r1), response of hotel booking request (r2)}</td>
</tr>
<tr>
<td>AR</td>
<td>{airline booking request (a)}</td>
</tr>
<tr>
<td>HR</td>
<td>{hotel booking request (h)}</td>
</tr>
<tr>
<td>C</td>
<td>{charge (c), successful notification (s)}</td>
</tr>
</tbody>
</table>

Table II
PARTICIPANT SERVICES AND THEIR ALPHABETS

Step 2: Determining the local behaviours for every participant. Based on Step 1 the participant Customer C, say, is associated with the occurrence of events c and s. To determine its local behaviour, all the SBVR rules that contain a FT for Customer identified in Step 1 are considered. In this case, the SBVR rule involved is:

It is obligatory that the customer receives a charge for the booking request from the acme-travel precedes the customer receives a successful notification of the booking request from the acme-travel.

We have seen that "precedence" in a rule takes the form occurrence of event precedes occurrence of event. Hence, the local behaviour of participant C is described by the trace \(cs\). Similarly for the other participant services, as shown in Table III.

<table>
<thead>
<tr>
<th>Participant Service</th>
<th>Local Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>(br_1,r_2,br_2)</td>
</tr>
<tr>
<td>AR</td>
<td>a</td>
</tr>
<tr>
<td>HR</td>
<td>h</td>
</tr>
<tr>
<td>C</td>
<td>(cs)</td>
</tr>
</tbody>
</table>

Table III
LOCAL BEHAVIOUR OF EACH PARTICIPANT SERVICE

Step 3: Deriving the order of interactions across participants. For this step we need to know how to identify the immediately succeeding message (next), hence also how to identify the next rule that gets activated. This needs to be done in a way that also takes into account alternative (choice) and concurrent messages.

1) Look at all the LHS of all BB rules in Table I. Each will specify a message, e.g., \(in(BB, [msg, m], t_1)\) specifies \(m\) at \(t_1\). There are two possibilities:

- If only one LHS of a rule has such message \(m\), then the RHS of that rule gets activated and the corresponding message occurs.
- If more than one rules have the same message \(m\) on their LHS, then look at the time \(t_i\), for \(1 \leq i \leq n\). If the rules have identical \(t_i\), it means that there is a choice between the corresponding messages, a case which is handled under Alternative in the sequel; otherwise, choose the smaller \(i\) and the RHS of the associated rule gets activated. For example,

  R1: \(in(BB, [msg, r_1], t_3) \rightarrow in(BB, [msg, r_2], nin(bb, [msg, r_1])\)
  R2: \(in(BB, [msg, r_2], t_3) \rightarrow in(BB, [msg, r_1], nin(bb, [msg, r_2])\)
  R3: \(in(BB, [msg, r_2], t_4) \rightarrow in(BB, [msg, c])\)

The LHS of the bottom two rules R1 and R2 concern the same message. Thus, the rule R2 is chosen after the RHS of the first rule has been activated.

2) The RHS selected in 1) above can have:

   - one in primitive only;
   - more than one in primitive, in which case look at Parallel in the sequel;
   - more than one in primitive with one or more nin primitive. For example,

     R1: \(in(BB, [msg, r_1], t_3) \rightarrow in(BB, [msg, r_2], nin(bb, [msg, r_1])\)
     R2: \(in(BB, [msg, r_2], t_3) \rightarrow in(BB, [msg, r_1], nin(bb, [msg, r_2])\)
     R3: \(in(BB, [msg, r_2], t_4) \rightarrow in(BB, [msg, c])\)

   The purpose of nin primitive in the first rule is to ensure that the event \(r_1\) will not be executed again after \(r_2\) has occurred. This is to avoid infinite chains of repetitive messages (livelock).

3) After the RHS has fired, look for any LHS which is the same as that RHS. Apply 1) of this Step to determine the next message.

   - Sequential. The BB rules have a key role to play here. This is due to their ability to express immediate causality, denoted by \(\rightarrow\). For example,

     \(in(BB, [msg, c], t_5) \rightarrow in(BB, [msg, s])\)

   where LHS describes the occurrence of \(c\), \(\rightarrow\) : describes that the LHS message immediately precedes the RHS message, and RHS describes the occurrence of \(s\). The incremental time \(t_i\), for \(1 \leq i \leq n\) in the LHS of the rule specifies that the time for LHS of next rule (the one following the activation of the RHS of this rule) should be \(t_i\). Therefore, from this BB rule we derive that \(c\) immediately precedes \(s\). Step 1 tells us in addition that both messages occur on participant service C (Customer).
• **Parallel.** While SBVR rules do not explicitly express concurrency, the corresponding BB rules do. They use the \(\cdot\) in between two or more \(in\) primitives to denote that the corresponding messages are concurrent. For example,

\[
R_1: \text{in}(BB, [msg, b], t_1) \rightarrow \\
\text{in}(BB, [msg, a], \text{in}(BB, [msg, h]) \\
\text{and} R_2: \\
\text{in}(BB, [msg, a], \{t_{2,1} : t_{2,2}\}), \text{in}(BB, [msg, h], \{t_{2,1} : t_{2,2}\}) \rightarrow \text{in}(BB, [msg, r_1])
\]

From the first rule, R1, we may deduce that message \(b\) immediately precedes the occurrence of \(a\) and \(h\), which occur concurrently. The use of the interval time in the second rule, R2, shows that the messages \(a\) and \(h\) occur within the lower and upper bound of time \(t_2\).

• **Alternative (choice).** A choice between two events can be deduced from the BB rules by looking for rules in the rule set (Table I) that have the same LHS appearing with a same time \(t_i\) and a different RHS. For example,

\[
R_1: \text{in}(BB, [msg, a], \{t_{1,1} : t_{1,2}\}), \text{in}(BB, [msg, h], \{t_{2,1} : t_{2,2}\}) \rightarrow \\
\text{in}(BB, [msg, r_1]) \\
R_2: \text{in}(BB, [msg, a], \{t_{1,1} : t_{1,2}\}), \text{in}(BB, [msg, h], \{t_{2,1} : t_{2,2}\}) \rightarrow \\
\text{in}(BB, [msg, r_2])
\]

where the common LHS describes the occurrence of message \(a\) concurrently with message \(h\), \(\rightarrow\) denotes ‘immediately precedes’, and then there is a choice between the RHS of rule R1 describing the occurrence of \(r_1\) and the RHS of the second describing \(r_2\).

Since the common LHS specifies the same message, but appears in different rules with different RHSs, which must both be present on the blackboard \(BB\) at the same specified time, the result is a branch in the corresponding order structure. Hence, from the BB rules R2 and R2 above, we may deduce that ‘the concurrent execution of \(a\) and \(h\) immediately precedes the execution of either \(r_1\) or \(r_2\)’.

**Step 4: Generate the corresponding vector language.**

Vectors are tuples of sequences of actions, one for each participating instance (Section II). We have the set of messages associated with each participant from Step 1 above. Each participant service is assigned to a coordinate of the vector representation, e.g., \(AT\) is assigned to the first coordinate, \(AR\) to the second, \(HR\) to the third, and \(C\) to the fourth. We also have the order in which these events occur from Step 3. Using this information we can go through the order structure of the events (produced in Step 3) and populate the vector language, as shown in Fig. 3.

This is done using one vector for each message, starting from the empty vector, denoted by \((\Lambda, \Lambda, \Lambda, \Lambda)\) and considering each message in turn until we have gone through the chain of messages in Step 3 above. Each message appears in the coordinate where it is received, as shown in Fig. 3.

Finally, the verification that the rule-based specification will produce a multi-party conversation that conforms to the original choreography given in terms of the UML 2 sequence diagram can be performed by checking the vector language of Fig. 3 against the vector language of Fig. 2 (right). It can be seen that the two are identical, so the local behaviours of the four participant services conform to the global protocol defined by the choreography.

**V. RELATED WORK**

Most work to date on the coordination of distributed systems is concerned with orchestration, where it is more natural to assume a central coordinator (the *orchestrator*) that is responsible for invoking and combining the participant services’ contributions to meet a given business request.

Formal approaches that reason about correctness of service choreography, rather than orchestration, are fairly limited in comparison. The work in [5] is targeted at WS-CDL [1] and describes a semantics given in terms of a notion of structural congruence and this is interpreted over transition systems. Vector languages used in our approach extend to a certain type of automata studied in [27], which draw upon asynchronous transition systems [19] and thus can model true concurrency. The work in [30] describes a Finite State Process (FSP) algebra for modelling the behaviour of interacting services, but the notation describes the system behaviour as one access point and thus does not capture what happens across the choreography while the accompanying LTSA-WS tool targets WS-BPEL [2] implementations.

Other formal work includes [31] which targets the Web Services Choreography Interface (WSCI), a predecessor to WS-CDL where the behaviour of a participant is described in terms of the role it plays in the conversation. Hence, like [32] it talks about choreography but in fact it targets WS-BPEL [2] (orchestration).

The work in [33] provides a declarative language *DecSerFlow* and uses the SCIFF framework for reasoning about
choreographies. The approach we proposed here has similar objectives but SBVR and its structured English (SBVR-SE) is an OMG standard and can be understood by both humans and machines while DecSerFlow is a proprietary graphical modelling language.

VI. CONCLUDING REMARKS AND FUTURE WORK

We presented a declarative approach to choreography specification and its translation to a trace-based model, vector languages for verification. Focusing on what rather than how means that one can start with an as unconstrained view as possible and only gradually constrain it by adding rules as the business activity becomes more clear. The structured English serialisation of SBVR allows domain specialists to readily validate the specification themselves. The vector language generated from the SBVR and BB rules is used to verify whether the local behaviours of the participant services conform to the global protocol prescribed by the choreography.

The notation of vector languages respects the loose-coupling between services and can be used to model long-running transactions [22], [34] involving the execution of service chains [35], [36]. [37] proposes an alternative architecture to WS-* [14] for large-scale distributed applications built on RESTful services, and SBVR as a modelling language.

Another consequence of adopting this trace-based model is that choreography projection can be defined in structured operational semantics. The construction reminds of the projections which are used to define the semantics of parallel composition in COSY [38] and CSP, in the sense that if \( v = u_1 \parallel u_2 \), then \( u_1 \) can be ‘recovered’ by restricting \( v \) to \( I_1 \), and likewise for \( u_2 \).

It transpires that the rule set which features predominantly in the choreography specification in our approach would need to be checked for consistency, i.e., detect subsets of rules that do not produce a solution space due to conflict or non-satisfiability of the conjunction. SBVR comes with a Logical Formulation for the rules based around first-order logic. The work on the SBVR2Alloy compilation tool [4] builds on this. However, SBVR includes alethic and deontic modalities [14], hence FOL constraint solvers will not suffice. Recent work on FODAL [39] tackles consistency checking for SBVR rules. This concerns verification of the SBVR model and as such is complementary to verification of the choreography described here. This is a direction that deserves further investigation.

Using context-dependent rules in systems engineering facilitates further analysis. The rule-based approach paves the way for explainable decision control (e.g., [40]) applied in steering a system towards a desired outcome, which in turn can be useful in complex recommender systems [41].

Another dimension has to do with the transformation from natural language to SBVR [15]. Controlled Language (CL) is used in [16] to ensure correctness and deadlock-freedom. This would allow our approach to go beyond structured English specifications for specifying business activities and would perhaps appeal to an even wider business audience.

REFERENCES


