XML Methods for Validation of Temporal Properties on Message Traces with Data*

Sylvain Hallé[1] and Roger Villemaire[2]

[1] University of California, Santa Barbara
shalle@acm.org
[2] Université du Québec à Montréal
villemaire.roger@uqam.ca

Abstract. We perform trace validation of LTL formulæ by exclusively using readily-available XML technologies. We first provide a translation between LTL and a subset of the XML Query Language XQuery, and show that an efficient validation of LTL formulæ can be achieved through the evaluation of XQuery expressions. Moreover, since LTL maps to a small fragment of XQuery, we show that that extending it to LTL-FO$^+$, which includes full first-order quantification over message contents, can be supported without additional cost. All these capabilities, which come "for free" in any web service environment providing a standard XQuery engine, are tested using a representative choreography scenario.

1 Introduction and Related Work

Web service choreographies define collaboration protocols between cooperating web service peers which communicate through a set of messages. These protocols can specify the order in which some messages must be received or sent by the peers, or provide conditions under which a particular sequence of messages can appear. An actual execution of a collaboration between services produces a trace of messages that is called a *conversation*. Protocols such as WS-Coordination can be used at runtime to ensure that each participant respects a given protocol; however, such constraints can also be checked statically on a recorded message trace. When an external observer records a conversation between services, the process of looking for violations of a specific choreography on the trace of exchanged messages is called *trace validation*.

In recent years, the importance of specifying choreographies manifested itself in the development and use of numerous languages to define cooperations between services. Among them, we find Conversation Specifications [5], the Web Service Choreography Description Language (WS-CDL) developed by the W3C [18], the re-use of UML Sequence Diagrams [14] in a web service context [11], SSDL Message Exchange Patterns (MEP) [21], the *Let's Dance*

* This work was done when Sylvain Hallé was at Université du Québec à Montréal. We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada on this research.

R. Meersman and Z. Tari (Eds.): OTM 2008, Part I, LNCS 5331, pp. 337–353, 2008.

choreography description language [9], and the Web Service Choreography Interface (WSCI) [1].

Linear Temporal Logic (LTL) has also been suggested for the expression of choreography constraints. [20] uses LTL to express behavioural constraints on the execution of a web service. Multi-agent web services are modelled in [23] using a custom protocol language called MAP which is then translated into SPIN models and model-checked with LTL formulæ. In [10], model checking of LTL formulæ expressed in Promela on BPEL processes is attempted using SPIN. Similarly, many approaches use equivalent, finite automata methods to express protocol specifications. For example, [19] models web service compositions by finite-state systems. An extension of LTL called LTL-FO$^+$ is presented in [15] to express choreography constraints that correlate the sequentiality of the messages and their data content.

However, although these initiatives provide structured, expressive languages to describe a variety of choreographies, the question of actually enforcing these choreographies on web services in production environments remains open. Most of these works provide specific algorithms for checking a trace of messages. These algorithms, though, are not part of production environments and have to be implemented. In our view, leveraging existing technologies and providing ways to minimize the changes required to a web service environment is a critical factor in the widespread adoption of validation techniques by providers and users.

In this paper, we study the validation of web service choreographies on XML message traces from the angle of the XML processing languages XPath and XQuery. The use of XML methods for validation of *temporal* message exchange patterns has been little studied. The only work we are aware of is [22] which translates Hoare's event-condition-action and request-response patterns into XQuery expressions; however, the translation is much more complex than the one we provide here, and does not apply for full-fledged temporal logic operators.

In Section 2, we recall web service choreographies and provide examples of choreography constraints based on a typical web service scenario. We show in Section 3 how these constraints can be expressed in LTL, or in an extension allowing quantification on data content called LTL-FO$^+$. In Section 4, we argue for the use of XML methods to validate these formulæ on message traces. We provide a mapping from LTL to a subset of XQuery called XPath 2.0 and perform an empirical validation of our approach by checking traces of more than 1,500 messages with the Saxon XQuery engine. Moreover, we show that the more expressive LTL-FO$^+$ can also be translated into XPath 2.0. This shows that trace validation of web service choreographies can be done using technologies already present in web service environments.

2 Web Service Choreographies

Web services interact between each other through the transmission of data units in the form of SOAP messages, whose general structure is defined by WSDL specifications. Numerous use cases arguing for web service choreography

specifications have been presented in the literature. To illustrate these concepts, we shall present one of them and show examples of choreography constraints.

2.1 An Example

Let us consider the case of an online trading company, adapted from [17]. A trading company is responsible for selling and buying stocks for its customers; it exposes its functionalities through a web service interface. An external buyer (which can be a human interfacing through a web portal, or another web service acting on behalf of some customer) can communicate through a set of XML messages, each representing a business operation performed by the trading company. This general context is appropriate to represent the basic requirements of e-commerce applications. Figure 1 shows the set of business operations available.

At first, a customer can connect to the trading company and ask for the list of available products. This is done through the exchange of a getAllStocks message, to which the company replies with a stockList message. The customer can then decide to get more information about each stock, such as its price and available quantity, using a getStockDetails message giving the list of stock names on which information is asked:

```
<message>
  <action>getStockDetails</action>
  <stocks>
    <stock-name> s_1 </stock-name>
    ...
    <stock-name> s_n </stock-name>
  </stocks>
</message>
```

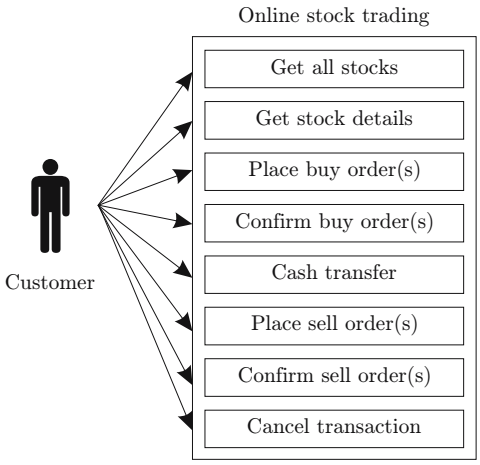

Fig. 1. Messages exchanged with the online trading company

The trading company replies with a stockDetails message listing the information of each stock:

```
<message>
  <action>stockDetails</action>
  <stocks>
    <stock>
      <name> s_1 </name>
      <price> p_1 </price>
    </stock>
    ...
    <stock>
      <name> s_n </name>
      <price> p_n </price>
    </stock>
  </stocks>
</message>
```

Finally, the customer can buy or sell stock products. In the case of a buy, this is done by first placing a placeBuyOrder message of the following form, listing the name and desired amount of each products to be bought:

```
<message>
  <action>placeBuyOrder</action>
  <stock>
    <name> s_1 </name>
    <amount> a_1 </amount>
  </stock>
    ...
  <stock>
    <name> s_1 </name>
    <amount> a_1 </amount>
  </stock>
</message>
```

The trading company checks the availability of each product and returns a confirmation of the transaction giving a bill identifier:

```
<message>
  <action>placeBuyOrderConfirm</action>
  <bill-id> b </bill-id>
</message>
```

The last step is for the customer to complete the transaction by proceeding to a cash transfer. This is achieved by providing an account number:

<message>
<action>cashTransfer</action>
<account> a </account>
<bill-id> b_1 </bill-id>
...
<bill-id> b_n </bill-id>
</message>

The transfer can be done for multiple buy orders at the same time. Alternatively, instead of a cash transfer, a cancelTransaction message listing some bill-ids can be sent to revoke these transactions before payment. A sell operation works similarly.

All these operations can be intertwined. At any moment, a customer can ask information about particular products, place concurrent buy/sell orders, complete or cancel any number of pending transactions.

2.2 Choreography Constraints on Message Traces

During a transaction between web services, several message traces can be obtained:

- An external observer can record the global pattern of messages sent by each service and align them in the order in which they are intercepted. This global trace has been called in [5] a *conversation*.
- A service can record the messages that are received or sent from its ports. There exists one such local message trace for each service cooperating in a transaction.

The WSDL specification for the online trading company does not prevent messages from being called in any sequence, as web services were initially designed as stateless. However, as was pointed out in [17], it is well-known that not all message traces represent valid transactions, i.e. interactions between services that lead to the successful completion of the cooperation. An increasing number of web services operate in a *stateful* manner, and conditions ensuring a safe and sound cooperation between services must therefore be specified at the choreography level.

Constraints without Data. For example, a user can call a cash transfer by sending a cashTransfer message without having first specified whether a stock should be bought or sold, or which precise stock is concerned by the transaction. A first choreography constraint would then be:

Choreography Specification 1. *No cash transfer can be initiated before a buy or sell confirmation has been issued.*

The validation of such a constraint assumes that a global trace file be partitioned into independent "sessions" for each client interacting with the online trading

company. It is straightforward to imagine additional, similar sequencing dependencies between invoked operations. For example, in order to force a buyer to get the most up-to-date prices on stocks, the trading company might require that no stock order can be placed before the user asks for stock details at least once. This leads to a second choreography constraint:

Choreography Specification 2. *No buy order can be placed before some stock information has been asked.*

A guarantee on the termination of pending transactions can also be imposed. More precisely, the system can require that all transactions eventually complete in two possible ways:

Choreography Specification 3. *Once a buy/sell order has been placed, no other buy/sell order can be placed until the first one is completed by a payment or a cancellation.*

This last constraint is imperfect, since a same cash transfer can "unlock" both a buy and a sell order. Moreover, if multiple transactions can be cancelled or paid in a same message, it is not clear that this constraint will always represent the desired behaviour.

Data-Aware Constraints. Choreography Specification 3 requires the constraints to go beyond a mere sequencing of operations and to express correlations on the actual content of the messages exchanged between the buyer and the company. A more precise version of Choreography Specification 3 could be the following:

Choreography Specification 4. *Every buy/sell order is eventually completed by a cash transfer or a cancellation referring to the same bill ID.*

By referring to "every buy and sell order", this new constraint requires that each bill ID appearing in a confirmation eventually appears inside a cancellation or a cash transfer. It implies an access to the data content of the message (the bill ID), and moreover correlates data values at two different moments along the message trace.

This need to access and correlate the data content in multiple messages is not an exception and appears in many other natural properties. For example:

Choreography Specification 5. *No cash transfer can occur for a buy order that has previously been cancelled.*

Indeed, to make sure that a cash transfer occurs for a previously cancelled transaction, one is required to correlate the bill-id of two different messages.

These last two choreography specifications are called "data-aware", because the sequence of messages and their content are interdependent. The reader is referred to [16] for a deeper exposition of data-aware constraints.

Additional constraints can complexify the monitoring process: asynchronous communications, lost, delayed or out-of-order messages can distort an otherwise valid interaction. These issues are out of the scope of the present paper.

3 LTL Choreography Specifications

We now briefly recall how Linear Temporal Logic (LTL) can be used to formally express the previous choreography requirements. LTL has been introduced to express properties about states and sequences of states in systems called Kripke structures [8]. In the present case, the states to be considered are messages inside a conversation. Formally, let us denote by M the set of XML messages. A sequence of messages in M is called a message trace:

Definition 1 (Message trace). *A* message trace σ *is a sequence of messages* $\sigma_1\sigma_2\ldots$, *where* $\sigma_i \in M$ *for every* $i \geq 1$.

We write σ_i to denote the i-th message of the trace σ, and σ^i to denote the trace obtained from σ by starting at the i-th message.

The ground terms of LTL formulæ are Boolean expressions on states. In the present case, these ground terms are assertions about individual messages. We therefore define a domain function, which is used to fetch and compare values inside a message. This domain function receives an argument π representing a *path* from the root to some element of the message. This path is defined using standard, XPath 1.0 expressions.

Definition 2 (Domain function). *Let* D *be a domain of values, and* Π *be the set of path expressions. A domain function* $Dom_m(\pi)$ *is an application* $M \times \Pi \rightarrow 2^D$ *which, given a message* $m \in M$ *and a path* $\pi \in \Pi$, *returns a subset of* D.

For example, if we let Π be the set of XPath formulæ, $\pi \in \Pi$ be the particular formula "/message/stock-name", and $m \in M$ be the following message:

```
<message>
  <name>placeBuyOrder</name>
  <stock-name>stock-1</stock-name>
  <stock-name>stock-2</stock-name>
</message>
```

then $Dom_m(\pi) = \{\text{stock-1}, \text{stock-2}\}$.

LTL's syntax is based on classical propositional logic, using the connectives $\neg$ ("not"), $\vee$ ("or"), $\wedge$ ("and"), $\rightarrow$ ("implies"), to which four temporal operators have been added. An LTL formula is a well-formed combination of these operators and connectives, according to the usual construction rules:

Definition 3 (LTL syntax). *Let* π *be a formula in some language, and* $S \subseteq D$ *be a subset of* D. *Then we have the following:*

1. $\pi = S$ *is an LTL formula*
2. *If* φ *and* ψ *are LTL formulæ, then* $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, $\mathbf{G}\,\varphi$, $\mathbf{F}\,\varphi$, $\mathbf{X}\,\varphi$, $\varphi\,\mathbf{U}\,\psi$ *are LTL formulæ.*

Boolean connectives have their usual meaning. The temporal operator **G** means "globally". For example, the formula $\mathbf{G}\,\varphi$ means that formula φ is true in every message of the trace, starting from the current message. The operator **F** means "eventually"; the formula $\mathbf{F}\,\varphi$ is true if φ holds for some future message of the trace. The operator **X** means "next"; it is true whenever φ holds in the next message of the trace. Finally, the **U** operator means "until"; the formula $\varphi\,\mathbf{U}\,\psi$ is true if φ holds for all messages until some message satisfies ψ. The semantics below formalizes these definitions:

Definition 4 (LTL semantics). *Let φ be a LTL formula, π and S be defined as previously. We say that a message trace σ* satisfies *φ, and write $\sigma \models \varphi$, if and only if it respects the following rules:*

$$\begin{aligned}
\sigma &\models \pi = S \Leftrightarrow Dom_{\sigma_1}(\pi) \textit{ is equal to } S\\
\sigma &\models \neg\varphi \Leftrightarrow \sigma \not\models \varphi\\
\sigma &\models \varphi \vee \psi \Leftrightarrow \sigma \models \varphi \textit{ or } \sigma \models \psi\\
\sigma &\models \mathbf{X}\,\varphi \Leftrightarrow \sigma^2 \models \varphi\\
\sigma &\models \mathbf{G}\,\varphi \Leftrightarrow \sigma^i \models \varphi \textit{ for every } i \geq 1\\
\sigma &\models \varphi\,\mathbf{U}\,\psi \Leftrightarrow \textit{there exists } k \textit{ such that } \sigma^k \models \psi \textit{ and } \sigma^i \models \varphi \textit{ for every } i < k
\end{aligned}$$

As usual, we define $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$, $\mathbf{F}\,\varphi \equiv \neg\mathbf{G}\,\neg\varphi$ and $\pi \neq S \equiv \neg(\pi = S)$. We extend the notation and allow to write $\pi = d$ when $Dom_m(\pi) = \{d\}$.

Equipped with LTL, the choreography constraints of the previous section can be revisited and expressed in this formal language. For example, Choreography Property 1 becomes the following LTL Choreography Specification:

LTL Choreography Specification 1

$$\begin{aligned}
&(\mathbf{G}\ \textit{message/name} \neq \textit{cashTransfer}) \vee (\textit{message/name} \neq \textit{cashTransfer}\\
&\mathbf{U}\ (\textit{message/name} = \textit{placeBuyOrder} \vee \textit{message/name} = \textit{placeSellOrder})) \quad (1)
\end{aligned}$$

This formula explains that either no "cashTransfer" does not hold until either a message with name "placeBuyOrder" or with name "placeSellOrder" has been observed.

Similarly, properties 2 and 3 can be translated in LTL:

LTL Choreography Specification 2

$$\begin{aligned}
&(\mathbf{G}\ \textit{message/name} \neq \textit{buyOrder}) \vee (\textit{message/name} \neq \textit{cashTransfer}\\
&\qquad\qquad \mathbf{U}\ \textit{message/name} = \textit{getStockInfo}) \quad (2)
\end{aligned}$$

LTL Choreography Specification 3

$$\begin{aligned}
&\mathbf{G}\,(\textit{message/name} = \textit{placeBuyOrder} \rightarrow\\
&\qquad \mathbf{X}\,(\neg\textit{message/name} = \textit{placeBuyOrder}\,\mathbf{U}\\
&\qquad\quad (\textit{message/name} = \textit{cashTranfer}\\
&\qquad\qquad \vee\ \textit{message/name} = \textit{cancelTransaction}))) \quad (3)
\end{aligned}$$

We postpone the translation of properties 4–6 to the next section.

4 From LTL to XQuery

In this section, we show how LTL choreography constraints can be checked on message traces using XML methods. More precisely, we present a translation from LTL to XQuery that allows trace validation using XML query processors.

Three powerful languages on XML documents, standardized by the W3C, are currently available: the XML Path Language XPath version 1.0 [7] and 2.0 [3], and the XML Query Language [4]. These three languages represent increasing degrees of expressiveness, i.e.

$$\text{XPath } 1.0 \subset \text{XPath } 2.0 \subset \text{XQuery } 1.0$$

Therefore, an XPath 1.0 expression is also an XPath 2.0 expression, which itself is an XQuery 1.0 expression. Various environments provide XPath and XQuery; these languages are intended to become the standard way of accessing and processing XML-based data.

4.1 A Case for XML Methods

Since the message traces considered are formed of XML documents, is is natural to think that properties on such traces can be validated using XML methods. Remark that a message trace $\sigma = \sigma_1\sigma_2\ldots$, defined as above, can be seen as an XML document by itself. It suffices to encapsulate each message sequentially into a global *trace* element as follows:

<trace>
  <message> σ_1 </message>
  <message> σ_2 </message>
  ...
</trace>

Since child elements in XML documents are ordered, the sequential ordering of the messages in σ is carried through the sibling ordering of the <message> elements. This property can be used to map temporal operators in LTL into sibling-ordering functions in an XML processing language. We list a few advantages of doing so:

1. XML query processors are common. Many existing solutions for trace validation require changes to the web service execution environments. For example, runtime monitors in [2, 16] require the implementation of their own monitoring algorithms. In contrast, most (if not all) web service execution environments provide XML processing capabilities natively; hence, translating LTL into XML query evaluation allows to perform trace validation without any new technology. Using XML query engines to perform trace validation is a powerful way of leveraging existing resources available in production environments while minimizing the modifications required to a service.

2. XML query languages are standardized. This entails that the validation of a trace need not be expressed in LTL: as long as it can be translated into an XML query expression, any standard implementation of the query engine will be able to process it.
3. XML query processors are efficient. Experimental results shown later in this section indicate that, in contexts where data domains of 100 elements and transactions formed of up to 1,500 messages, the validation of a trace requires less than 10 ms per message.

It remains to determine which, if any, of the three XML languages XPath 1.0/2.0 or XQuery 1.0, is the most appropriate for the task.

Validating a trace of events against an LTL formula φ amounts to checking whether a particular word is accepted by a Büchi automaton built from φ; according to this classical result, the size of this automaton is exponential in the length of φ [12]. This contrasts with the complexity of checking a document against an XPath expression, which has been shown to be P-complete [13]. From these two results, the following conclusion can be drawn:

Theorem 1. *XPath 1.0 cannot be used for LTL trace validation.*

Indeed, XPath 1.0 lacks the quantification mechanism that is required for the translation. However, XQuery 1.0, as well as its subset XPath 2.0, are Turing-complete [13]. Therefore, both are powerful enough to model LTL. The main difference between the two is the presence in XQuery of a powerful node selection mechanism called *FLWOR expressions*. Since FLWOR expressions are not necessary for the translation shown below, the required language is therefore XPath 2.0. In the following, we nonetheless call our formulæ "XQuery expressions" to avoid confusion with XPath 1.0; the reader must keep in mind that our translation actually belongs to the stricter subset XPath 2.0.

4.2 Translation to XQuery

Based on the previous observation, we develop a recursive translation function ω_ρ, which takes as input an LTL formula and produces an equivalent XQuery expression. The function also carries a parameter ρ, which is a pointer to the root of the first message of the current trace. When starting the translation, ρ must point to the first message of the trace document. The XPath expression `/*/*[1]` can be used to designate this first message: it points to the first element under the root element.

It then suffices to define ω_ρ for each LTL operator. The translation of XML path expressions is direct: for π a path in a message and $d \in D$ a value at the end of that path, we have:

$$\omega_\rho(\pi = d) \equiv \rho/\pi = d \tag{4}$$

It is important to remark that the path π is *relative* to the current message of the trace; hence π must be appended to ρ. Since XQuery allows all logical connectors, the translation of $\neg$, $\vee$ and $\wedge$ is also straightforward:

$$\omega_\rho(\neg\varphi) \equiv \mathsf{not}\ (\omega_\rho(\varphi)) \tag{5}$$
$$\omega_\rho(\varphi \vee \psi) \equiv (\omega_\rho(\varphi)\ \mathsf{or}\ \omega_\rho(\psi)) \tag{6}$$
$$\omega_\rho(\varphi \wedge \psi) \equiv (\omega_\rho(\varphi)\ \mathsf{and}\ \omega_\rho(\psi)) \tag{7}$$
$$\omega_\rho(\varphi \rightarrow \psi) \equiv (\mathsf{not}(\omega_\rho)(\varphi)\ \mathsf{or}\ \omega_\rho(\psi)) \tag{8}$$

It remains to translate the temporal operators into equivalent XQuery code. We consider first the case of the **G** operator. According to the semantics of LTL, a formula of the form $\mathbf{G}\,\varphi$ is true on the trace which starts at the current message, if and only if all subsequent messages (including the current one), satisfy φ. Since ρ is a pointer to the current message, then the XPath expression ρ/following-sibling::* denotes all the messages following ρ. We perform the union of this set with ρ to obtain the desired messages: ρ union ρ/following-sibling::*. The XQuery formula must then express that each message in this set satisfies the remaining formula φ, or more precisely, the translation of φ into XQuery. We obtain the following expression:

$$\omega_\rho(\mathbf{G}\,\varphi) \quad \equiv \quad \mathsf{every}\ \$\mathsf{x}\ \mathsf{in}\ (\rho\ \mathsf{union}\ \rho/\mathsf{following\text{-}sibling::*})\ \mathsf{satisfies}\ \omega_{\$\mathsf{x}}(\varphi) \tag{9}$$

In this translation, $x is a fresh XQuery variable, bound to the every statement, that is introduced to designate successively each message in the set of desired messages. Remark that since φ must be true on each such message, the root on which of φ is evaluated is $x.

The translation of the "next" (**X**) operator, can be seen as a special case of **G**, where the set of desired states contains only the immediate successor to ρ. This candidate set can be obtained by the XPath expression $\rho' = \rho$/following-sibling::*[1]: the first message following the one pointed by ρ. The rest of the translation is identical:

$$\omega_\rho(\mathbf{X}\,\varphi) \equiv \mathsf{every}\ \$\mathsf{x}\ \mathsf{in}\ (\rho/\mathsf{following\text{-}sibling::*[1]})\ \mathsf{satisfies}\ \omega_{\$\mathsf{x}}(\varphi) \tag{10}$$

One can see that the translation to XQuery, although yielding a more verbose expression, closely sticks to the original semantic definition for LTL. We omit the details for the remaining temporal operators and give directly their translation pattern into XQuery:

$$\omega_\rho(\mathbf{F}\,\varphi) \equiv \mathsf{some}\ \$\mathsf{x}\ \mathsf{in}\ (\rho\ \mathsf{union}\ \rho/\mathsf{following\text{-}sibling::*})\ \mathsf{satisfies}\ \omega_{\$\mathsf{x}}(\varphi) \tag{11}$$

$$\omega_\rho(\varphi\,\mathbf{U}\,\psi) \equiv \mathsf{some}\ \$\mathsf{x}\ \mathsf{in}\ (\rho\ \mathsf{union}\ \rho/\mathsf{following\text{-}sibling::*})\ \mathsf{satisfies}\ \mathsf{every}\ \$\mathsf{y}\ \mathsf{in}\ (\$\mathsf{x}/\mathsf{previous\text{-}sibling::*}\ \mathsf{intersect}\ (\rho\ \mathsf{union}\ \rho/\mathsf{following\text{-}sibling::*}))\ \mathsf{satisfies}\ \omega_{\$\mathsf{y}}(\psi) \tag{12}$$

As an example, applying ω to LTL Choreography Specification 2 yields the following XQuery expression:

```
(every $_0 in /*/*[1] union /*/*[1]/following-sibling::* satisfies
  not($_0/name = "placeBuyOrder"))
or
(some $_0 in /*/*[1] union /*/*[1]/following-sibling::* satisfies
  (every $__0 in (/*/*[1] union /*/*[1]/following-sibling::*)
    intersect ($_0/preceding-sibling::*) satisfies
      not($__0/name = "placeBuyOrder"
        and $__0/name = "getStockDetails")))
```

An interesting consequence of this mapping is that any other notation translatable into LTL can also be validated using XPath. This includes UML Sequence Diagrams [14] (or more specifically Message Sequence Charts), since various algorithms to translate them into LTL have been developed, for instance in [6,11]. This also includes SSDL's Message Exchange Patterns (MEP) and Rules protocol frameworks [21], and the *Let's Dance* choreography description language [9].

4.3 Support for LTL-FO$^+$

Up to now, Choreography Specifications 4 and 5 have been left out of the discussion. These constraints require a correlation between data content of multiple messages in the trace and have been dubbed "data-aware" for that reason. However, traditional temporal logics such as LTL are not expressive enough to model these constraints, as has been shown in [16].

An extension to LTL called LTL-FO$^+$ is required. This extension allows a first-order quantification on the content of messages; it was introduced in [15] and is defined as follows:

Definition 5 (LTL-FO$^+$ syntax). *Let $\pi \in \Pi$ be defined as previously. We have the following:*

- *Any LTL formula is an LTL-FO$^+$ formula*
- *If φ is a LTL-FO$^+$ formula, x_i is a free variable in φ, $\pi \in \Pi$ is a path expression, then $\exists_\pi x_i : \varphi$ and $\forall_\pi x_i : \varphi$ are LTL-FO$^+$ formulæ.*

Definition 6 (LTL-FO$^+$ semantics). *Let φ be a LTL-FO$^+$ formula, π and S be defined as previously. We say that a message trace σ* satisfies *φ, and write $\sigma \models \varphi$, if and only if it respects the rules in Definition 4, plus the following rules:*

$$\sigma \models \exists_\pi x : \varphi \Leftrightarrow \sigma \models \varphi[x/k] \text{ for some } k \text{ in } Dom_{\sigma_1}(\pi)$$
$$\sigma \models \forall_\pi x : \varphi \Leftrightarrow \sigma \models \varphi[x/k] \text{ for every } k \text{ in } Dom_{\sigma_1}(\pi)$$

Properties 4–6, which require an extension to LTL to include first-order quantification, can now be translated using this extension.

LTL-FO$^+$ Choreography Specification 4

$$\mathbf{G}\,(\mathit{message/name} = \mathit{placeBuyOrderConfirm} \rightarrow \forall_{\mathit{message/bill\text{-}id}}\, x : \mathbf{F}\,((\mathit{message/name} = \mathit{cancelTransaction} \vee \mathit{message/name} = \mathit{cashTransfer}) \wedge \exists_{\mathit{message/bill\text{-}id}}\, y : x = y)) \quad (13)$$

This property reads as follows: globally along the whole message trace, whenever a placeBuyOrderConfirm message is observed, the following holds: for every bill-id x inside that confirmation message, eventually in the future, a cancelTransaction or a cashTransfer message is be observed; moreover, there exists a bill-id y inside that message which is the same as x. This indeed expresses the requirement that every buy/sell order is eventually completed by a cash transfer or a cancellation. Remark that universal and existential quantifiers on message content are needed to express this specification.

Similarly, Choreography Specification 5 can be expressed as the following LTL-FO$^+$ formula:

LTL-FO$^+$ Choreography Specification 5

$$\mathbf{G}\,(\mathit{message/name} = \mathit{cancelTransaction} \rightarrow \forall_{\,\mathit{message/bill\text{-}id}}\, x : \mathbf{G}\,\neg(\mathit{message/name} = \mathit{cashTransfer} \wedge \forall_{\,\mathit{message/bill\text{-}id}}\, y : x = y)) \quad (14)$$

An interesting question is to determine whether the mapping from LTL to XPath 2.0 can be extended to include LTL-FO$^+$. The result, positive, follows from the Turing-completeness of the language. Indeed, it suffices to add the following two mapping rules to the original ω to support LTL-FO$^+$'s quantifiers on data using XPath 2.0:

$$\omega_\rho(\exists_\pi x) = \text{ some \$x in } \rho/\pi \text{ satisfies } \omega_\rho(\varphi) \quad (15)$$

$$\omega_\rho(\forall_\pi x) = \text{ every \$x in } \rho/\pi \text{ satisfies } \omega_\rho(\varphi) \quad (16)$$

To follow XQuery's syntactical requirements for variables, every occurrence of the quantified variable x in φ must be prefixed by a \$.

This result is noteworthy. It shows that performing trace validation on an extension of the temporal logic LTL that includes full first-order quantification over message contents can be supported without any additional modification by existing XML processing engines.

4.4 Experimental Results

To support these claims, we performed an experimental evaluation of the method on automatically-generated traces taken from the stock trading company example described in Section 2. The goal of these experiments was to show that validating choreography constraints expressed in LTL and LTL-FO$^+$ on XML message traces can be effectively done using an XML processing engine.

To this end, we produced a set of 100 trace files of length ranging from 10 to approximately 1,500 messages, each containing a randomly created sequence of the messages described in the example: stock products are displayed, information for some stocks is retrieved, and an arbitrary number of buys, sells, payments and cancellations occurs in various orderings. Each individual message could refer to a maximum of 100 different stocks.

Each of these traces was then sent to an XQuery engine, which evaluated the XQuery translation of each Choreography Specifications 1–5. The total running time for evaluating each formula on each trace was then measured. Scatterplots of the results are shown in Figure 2.

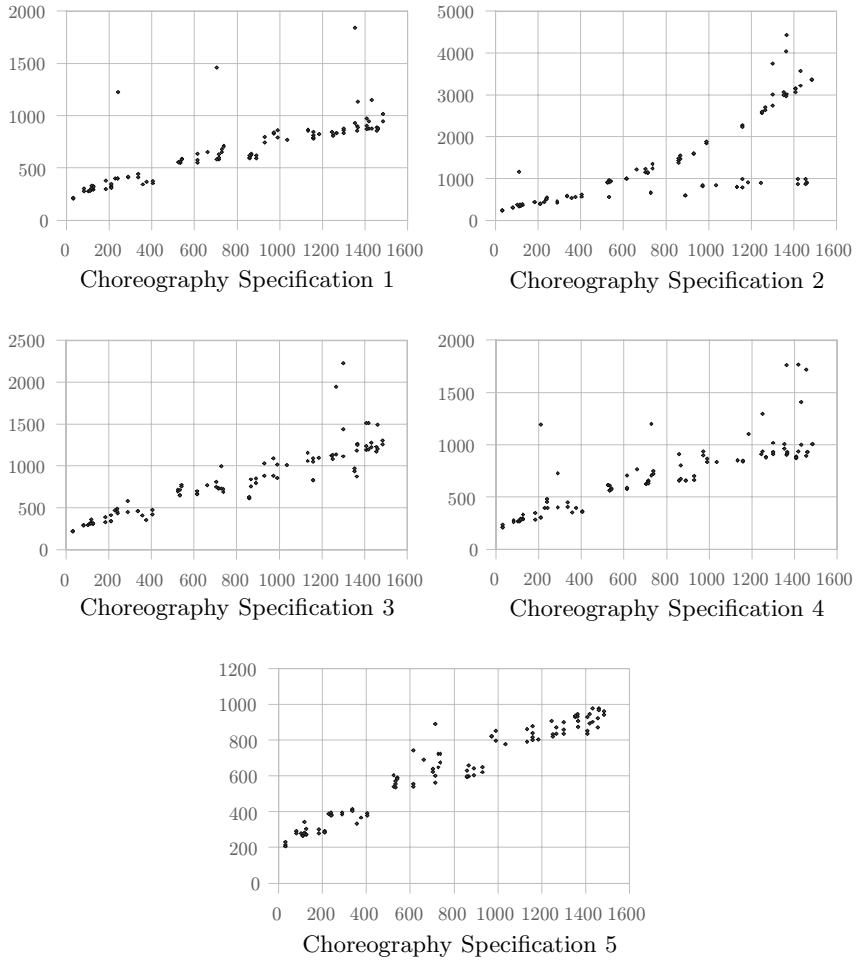

Fig. 2. Scatterplots of total validation time for Choreography Specifications 1–5. The x-axis plots the number of messages in the trace, and the y-axis plots the total validation time of the trace in milliseconds. The data domain is of size 100.

The XQuery engine chosen for the tests was Saxon-B 9.0 running on an AMD Athlon XP 2200+ system under Cygwin. Saxon-B is a mature, industry-level, and freely available engine.[1] It was not modified in any way. A simple

[1] http://www.saxonica.com/

front-end was coded to automatically translate LTL(-FO$^+$) formulæ into XQuery. Translation times are negligible.

One can observe that barring exceptional cases, the average time required to process one message does not exceed 5 milliseconds; it never went above 10 milliseconds for all traces. There seems to be, however, a fixed cost of around 200 ms to invoke the XQuery engine, no matter the length of the trace; therefore the highest processing times per message have been encountered on the smallest traces.

Processing times grow in roughly the same fashion for each of the five formulæ. An interesting point is that the total processing time remains similar between the group of Constraints 1–3, which are plain LTL formulæ, and the group of Constraints 4–5, which require LTL-FO$^+$'s first-order quantification mechanism to be expressed. This indicates that, for the Choreography Specifications studied in these experiments, extending trace validation to temporal first-order properties added no noticeable load on the XML query engine, and can therefore be performed "for free", using the same XML technologies that are required for the more traditional LTL.

5 Conclusion

In this paper, we have shown how representative web service choreography constraints can be checked on traces of messages using XML methods. In particular, we showed how any formula in the Linear Temporal Logic LTL can be translated into an equivalent XQuery expression on a message trace. This allows standard, off-the-shelf XML query processors such as Saxon to validate choreography constraints. Moreover, we have shown that extending the language to LTL-FO$^+$, which includes full first-order quantification over message contents, can be supported without additional cost. Since the translation takes a temporal logic as an input, any other language translatable into LTL or LTL-FO$^+$ can directly tap onto this method. This includes, among others, UML Sequence Diagrams. Finally, we showed how the validation of representative choreography constraints, expressed both in LTL and LTL-FO$^+$, can be done efficiently using an XQuery processor. We hope these results will open the way to other works leveraging the power of XML query processors available in web service execution environments for the formal validation of collaborations between peers.

References

1. Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I., Zimek, S.: Web service choreography interface (WSCI) 1.0 (2002), `http://www.w3.org/TR/wsci`
2. Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run-time monitoring of instances and classes of web service compositions. In: ICWS, pp. 63–71. IEEE Computer Society, Los Alamitos (2006)

3. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Siméon, J.: XML path language (XPath) version 2.0, W3C recommendation (2007), http://www.w3.org/TR/xpath20
4. Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML query language, W3C working draft (2005), http://www.w3.org/TR/xquery/
5. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: a new approach to design and analysis of e-service composition. In: WWW, pp. 403–410 (2003)
6. Caporuscio, M., Inverardi, P., Pelliccione, P.: Compositional verification of middleware-based software architecture descriptions. In: ICSE, pp. 221–230. IEEE Computer Society, Los Alamitos (2004)
7. Clark, J., DeRose, S.: XML path language (XPath) version 1.0, W3C recommendation (1999), http://www.w3.org/TR/xpath
8. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)
9. Decker, G., Zaha, J.M., Dumas, M.: Execution semantics for service choreographies. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 163–177. Springer, Heidelberg (2006)
10. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E. (eds.) WWW, pp. 621–630. ACM, New York (2004)
11. Gan, Y., Chechik, M., Nejati, S., Bennett, J., O'Farrell, B., Waterhouse, J.: Runtime monitoring of web service conversations. In: CASCON 2007: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research, pp. 42–57. ACM, New York (2007)
12. Gerth, R., Peled, D., Vardi, M.Y., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: Dembinski, P., Sredniawa, M. (eds.) PSTV. IFIP Conference Proceedings, vol. 38, pp. 3–18. Chapman & Hall, Boca Raton (1995)
13. Gottlob, G., Koch, C., Pichler, R.: The complexity of xpath query evaluation. In: PODS, pp. 179–190. ACM, New York (2003)
14. Object Management Group. UML specification version 1.1, OMG document ad/97-08-11, http://www.omg.org/cgi-bin/doc?ad/97-08-11
15. Hallé, S., Villemaire, R.: Runtime monitoring of message-based workflows with data. In: EDOC (to appear, 2008)
16. Hallé, S., Villemaire, R., Cherkaoui, O., Ghandour, B.: Model-checking data-aware temporal workflow properties with CTL-FO+. In: EDOC, pp. 267–278. IEEE Computer Society, Los Alamitos (2007)
17. Josephraj, J.: Web services choreography in practice (2005), http://www-128.ibm.com/developerworks/webservices/library/ws-choreography/
18. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: Web services choreography description language version 1.0 (2005), http://www.w3.org/TR/ws-cdl-10/
19. Kazhamiakin, R., Pistore, M., Santuari, L.: Analysis of communication models in web service compositions. In: Carr, L., De Roure, D., Iyengar, A., Goble, C.A., Dahlin, M. (eds.) WWW, pp. 267–276. ACM, New York (2006)
20. Nakajima, S.: Lightweight formal analysis of web service flows. Progress in Informatics (2), 57–76 (2005)

21. Parastatidis, S., Webber, J., Woodman, S., Kuo, D., Greenfield, P.: SOAP service description language (SSDL). Technical Report CS-TR-899, University of Newcastle, Newcastle upon Tyne (2005)
22. Venzke, M.: Specifications using XQuery expressions on traces. Electr. Notes Theor. Comput. Sci. 105, 109–118 (2004)
23. Walton, C.D.: Model checking multi-agent web services (2004)