Model-checking Data-Aware Temporal Workflow Properties with CTL-FO$^+$

Sylvain Hallé, Roger Villemaire, Omar Cherkaoui and Boubker Ghandour
Université du Québec à Montréal
C. P. 8888, Succ. Centre-Ville
Montréal, Canada H3C 3P8
halle@info.uqam.ca [*]

Abstract

Most works that extend workflow validation beyond syntactical checking consider constraints on the sequence of messages exchanged between services. However, these constraints are expressed only in terms of message names and abstract away their actual data content. Using the context of the User-controlled Lightpath initiative (UCLP) hosted by the CANARIE consortium, we provide examples of real-world "data-aware" web service constraints where the sequence of messages and their content are interdependent. We present CTL-FO$^+$, an extension over Computation Tree Logic that includes first-order quantification on state variables in addition to temporal operators. We show how CTL-FO$^+$ is adequate for expressing data-aware constraints, give a complete model checking algorithm for CTL-FO$^+$ and establish its complexity to be PSPACE-complete. This makes using CTL-FO$^+$ for validating workflow properties no harder than using the Linear Temporal Logic (LTL) already used by some web service tools. Finally, we show how the modelling of data-aware properties is an increase in expressiveness that cannot be efficiently simulated by these tools.

1 Introduction

There exists a large number of web service orchestration tools available over the Internet; these tools allow a syntactical validation of the service invocations in a workflow, since all input and output messages are publicized by service providers in WSDL documents whose form and content is regulated by standards bodies such as the W3C. This "first generation" of web service technologies, as it is called by [33], concentrates on single request-response patterns of messages specified by various means such as Message Exchange Patterns (MEPs).

However, it has long been argued that syntactical correctness does not give a complete picture of the necessary conditions for a successful interaction with a service [24]. Nothing prevents a BPEL process from sending to a peer syntactically valid messages in a sequence that prevents an actual composition from taking place. This led the authors of [16] to call for future work on a formal language to express and advertise the *protocol* imposed on the use of a service, and a methodology to check as much as possible that an orchestration script created by some user satisfies this protocol before it is allowed to execute.

A "second generation" of web service technologies has given rise to a variety of standards taking into account the sequence of message exchanges allowed by a service. The SOAP Service Description Language (SSDL) [26] is a notable example of this approach. Classical temporal languages such as the Linear Temporal Logic (LTL), the Computation Tree Logic (CTL) or the π-calculus have been suggested as appropriate notations for expressing temporal or conversational dependencies between message exchanges. Numerous automated validation tools have also been developed that can guarantee conformance of a given workflow to some set of operating guidelines, in the spirit of [23].

Although this new generation of technologies allows for a much more realistic specification and enforcement of interaction constraints, most efforts still abstract the actual content that transits inside the messages of a given conversation. In other words, current protocol specifications treat messages as atomic units represented by their names; they are not "data-aware".

In this paper, we argue that data-awareness of protocol specifications is a fundamental part of ensuring workflow correctness. Using the context of the User-controlled Lightpath initiative (UCLP) hosted by the CANARIE consortium, we provide examples of real-world web service protocols where both the sequence of messages and their content are interdependent. We present an extension of the popular Computation Tree Logic (CTL) that introduces a general first-order quantification on state values, called CTL-FO$^+$,

[*]We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada on this research.

as an appropriate formal language for the expression of temporal constraints on web service invocations. Contrarily to the classical temporal formalisms used in most web service validation approaches, CTL-FO$^+$ retains the full temporal power of the CTL logic, while allowing to refer to the content of messages inside the temporal properties. We provide an explicit algorithm for the model checking of CTL-FO$^+$ formulæ on a given workflow model. We establish the complexity of the problem of model checking a CTL-FO$^+$ formulæ to be PSPACE-complete. This result places CTL-FO$^+$ model checking on a par, complexity-wise, with the Linear Temporal Logic (LTL) used by widely accepted tools like SPIN [17]. Therefore, we argue that data-awareness in web service validation is as tractable as modelling sequential properties in LTL, an approach that is already tackled by many second generation workflow tools.

The paper is structured as follows. In Section 2, we briefly review related work and show why current model checking solutions based on traditional temporal logics are not adequate for the validation task at hand. In Section 3, we describe the context of User-Controlled Lightpaths web services and show examples of data-aware constraints. In Section 4, we show how the use of CTL-FO$^+$ actually increases expressiveness to model richer web service constraints. A complete model checking algorithm is presented and its complexity is established in Section 5. Section 6 concludes and announces future work and improvements over the current methodology.

2 Related Work and Existing Solutions

The modelling and validation of constraints of various kinds on web service workflows has spawned a large amount of both theoretical and practical works. With respect to the goal of this paper, they can be classified into three categories corresponding to the degree of data-awareness they exhibit. We mention here a few of them.

To support our point, we illustrate each of these categories in the simple example of Figure 1. We consider a web service workflow which receives from some partner a message labelled “a” that contains an integer value. If this received value is 0, then the service returns a message “b” with value 9. If the received value is not 0, then the service returns a message “c” that increments the received value by 1. For the needs of the example, we employ a simplified notation to refer to messages sent (!a) or received (?a) and indicate the value of a message between parentheses. The $\gg$ symbol means “the next message”.

2.1 Propositional Workflows, Propositional Properties

A first step is to use classical automata-theoretic constructions or model checking tools and languages to model the behaviour of a web service and its interaction with other services. This is exemplified in Figure 1a. The messages are considered *atomic*: their actual data content is abstracted away. We call such a model *propositional*, since the external behaviour of web services is represented by the transmission or reception of messages that are identified by propositional letters standing for their names.

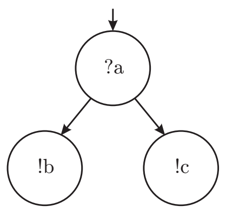
This entails that the choice between sending message “b” and message “c”, since it depends on message content, is seen as non-deterministic by the model. For the same reason, the behavioural properties of the service can only be expressed in terms of message names; we call them *propositional* properties. The two formulæ at the bottom of Figure 1a respectively mean that when message “a” is received, the next message is “b”, or that when message “a” is received, the next message is “c”.

A fair number of works use the propositional approach. Conversation specification [6] is an example of sequence of intertwined messages received and sent by multiple agents. Message Sequence Charts (MSC) are modelled into finite state processes by [12]. A similar approach has been done with use of the BPE-calculus and the Concurrency Workbench (CWB) in [21] and Petri nets in [29]. [33] tackles the formal specification of a protocol of interaction between services expressed as a pattern of messages.

These works have been dubbed by [10] “data-agnostic” solutions. It is important to note, however, that although these works do not model data, this abstraction is an appropriate simplification to tackle problems that are outside the scope of the present paper. For example, [33] provides an algorithm that determines whether services are “locally enforceable”; modelling the data content in messages in such a work is an open problem, and would render such a question much more complex and perhaps intractable in practice.

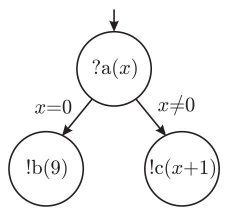
2.2 Data-aware Workflows, Propositional Properties

A refinement over the previous solutions is to consider that the actual data exchanged in the messages of a web service can actually influence the control flow of that service: the workflow model becomes “data-aware”. This refinement is illustrated in Figure 1b. The choice between sending message “b” or “c” is now unambiguous and determined by the value inside message “a”. Moreover, the model correctly represents that the value inside message “c” is always incremented by 1.



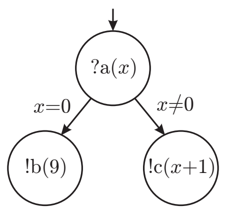
$?\mathrm{a} \gg !\mathrm{c}$
$?\mathrm{a} \gg !\mathrm{b}$

(a) Propositional workflow, propositional properties



$?\mathrm{a} \gg !\mathrm{c}$
$?\mathrm{a}(0) \gg !\mathrm{b}(9)$

(b) Data-aware workflow, propositional properties



$\forall x, y\ : ?\mathrm{a}(x) \gg (!\mathrm{c}(y) \wedge y = x + 1)$
$?\mathrm{a}(0) \gg !\mathrm{b}(9)$

(c) Data-aware workflow, data-aware properties

Figure 1. Workflow modelling with various degrees of data-awareness

This category constitutes the bulk of formal web services models. [20] models web service compositions by finite-state systems and studies them on the angle of synchronicity; it takes the content of variables and message parts into account by extending the original message alphabet. [3] models web services in Propositional Dynamic Logic (PDL) and is interested in generating automated compositions between services. [11] proposes a restricted BPEL semantics for which it is possible to automatically generate the composition of tasks. In [23], the controllability of a business process is studied; the *operating guidelines* of a process P is the automaton that includes as its subgraphs all the possible controllers of P. [25] proposes techniques to extract a behavioural specification from the BPEL program and to verify it with model checking techniques.

Other works also present automated tools for the validation of the properties. [30] formalizes BPEL web service workflows using a language called CHISEL which is then transformed into LOTOS for automated validation. Multi-agent web services are modelled in [32] using a custom protocol language called MAP which is then translated into SPIN models and model-checked. A process algebra approach is used in [5] to model web service choreographies using the Calculus of Communicating Systems (CCS). [27] uses a formal language called Tropos and validates properties in NuSMV. Finally, in [13], model checking of LTL formulæ expressed in Promela on BPEL specifications is attempted using SPIN. The approach is extended in [14] and constitutes the basis of the Web Service Analysis Tool (WSAT). VERBUS [2] is another tool that translates a web service workflow into a finite-state structure. Finally [19] studies the two-phase commit protocol and models it using the Temporal Logic of Actions (TLA$^+$).

Although these works take data into account when modelling the web services' interactions, this data does not play a role when expressing the properties. The temporal formulæ are still propositional. Actual data content can be referred to, but only statically by extending the original message alphabet. This is shown by the properties in Figure 1b. It is now possible to state that when "a" contains 0, then "b" is sent with value 9, since 0 and 9 are fixed constants: a(0) and b(9) are simply modelled as two new message names. It is not possible, however, to compare the values inside two different messages except by explicitly stating their value; therefore, one cannot say "for all x, the value inside message "a" is x, and later the value inside message "c" is $x + 1$" without resorting to explicitly name each possible static value. However, in most of these works the properties that need to be modelled do not require such a quantification.

2.3 Data-aware Workflows, Data-aware Properties

A further extension with respect to expressiveness of properties is to model the transfer and transformation of data inside the control flow of the modelled service, but also to allow quantification on data inside temporal properties. We call these properties "data-aware" to indicate that the actual message content can be known and fully accessed by the temporal formulæ. Figure 1c illustrates this. Knowledge about the internal workflow generally remains unchanged with respect to the previous category. However, the properties can now fully express the constraint between messages "a" and "c": when "c" is sent, it contains the value of "a" incremented by 1.

We are only aware of a limited number of works that tackle this problem. In [9, 10], extensions to the temporal logics CTL and LTL, respectively called CTL-FO and LTL-FO, are introduced. These logics include a form of first-order quantification on data. The model presented is very rich: it contains a *database* represented as a variable set of first-order predicates; however this richness is achieved at the price of complexity. The problem of model checking a

CTL-FO formula φ on a web service $\mathcal{W}$ (as defined in [9]) is undecidable. The problem of model checking a formula φ without any quantification is in CO-NEXPTIME if the formula is propositional CTL, and in EXPSPACE if the formula is propositional CTL*.

We show in this paper how a simpler modelling of the services, coupled with a more expressive logic than CTL-FO, is sufficient for model checking important data-aware properties in real-world scenarios. Theorem 1 will show that the logic CTL-FO$^+$ introduced in the present work extends CTL-FO in terms of expressiveness, while Theorem 2 will demonstrate that CTL-FO$^+$ model checking is PSPACE-complete, a considerably lower complexity.

Another work of interest is [31], which defines specifications using XQuery on traces (SXQT). The SOAP messages exchanged by services are aligned into a large XML sequence. XQuery can then be used to refer to and compare complex elements of specific messages along the trace in a powerful manner; temporal operators are translated into specific XQuery expressions. However, this approach allows the validation of one specific trace at a time and does not constitute a complete model checking of the service workflow itself.

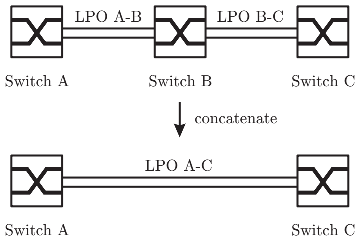


Figure 2. The result of the concatenation operation is an LPO that is considered as one single link.

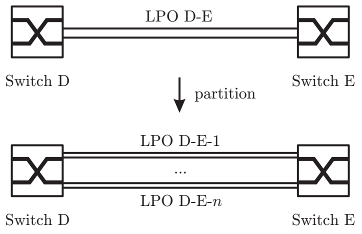


Figure 3. The partition operation splits an LPO into fragments of smaller bandwidth.

3 A Web Service Scenario

To measure the importance of data-awareness in web service workflow validation, we introduce a representative real-world scenario. We study this scenario under the angle of data constraints and show that data-aware properties arise naturally and are essential to correctly model and validate.

3.1 UCLP Web Service Architecture

To make a more efficient and flexible use of network resources, a growing trend is to offer users and applications services that can be reserved and composed according to specific needs. This is particularly visible in the GRID initiative which offers computing, data ware-housing and transmission resources for high-performance applications.

The web service paradigm is an ideal setting for such service-oriented networks. Based on this observation, the User-Controlled Lightpath (UCLP) research project initiated by the CANARIE Consortium[1] develops an environment that allows end users to self provision and dynamically reconfigure optical networks resources within a single domain or across multiple independent management domains. To this end, network resources from a specific provider are virtualized and exposed to the end user as instances of *web services* that implement functionalities related to lightpath manipulation. The project is currently at version 2.0. We concentrate in the following on the solution developed by the joint Université du Québec à Montréal-University of Ottawa UCLP development team[2], where such services are called Lightpath Objects (LPOs).

Simply put, a lightpath is a point-to-point, high-speed optical link. There are two main operations provided to manage Lightpath Objects.

LPO Concatenation. In order to build an end-to-end link, two adjacent LPOs can be *concatenated*, as is exemplified in Figure 2. The result of the concatenation operation is an LPO that is considered as one single link.

Since the same traffic will flow through all the link's segments, the concatenated LPOs must have the same bandwidth. Furthermore, the concatenation operation gives rise to a new LPO during whose lifetime the original LPOs cannot be used individually in some other operation.

LPO Partition. An LPO's bandwidth can be *partitioned* into links of equal bandwidth. For instance an OC-3 LPO (155.52 Mbps) can be partitioned into three OC-1 LPOs (51.84 Mbps). This is shown in Figure 3.

In order to be partitioned into OC-1 links, an LPO must be of an OC-1's multiple bandwidth. Furthermore, as before, during the partition's life-time the original LPO cannot be used in other operations.

[1]http://www.canarie.ca/canet4/uclp/

[2]The software developed by all CANARIE funded development teams is freely available from their site http://www.uclpv2.ca.

Within the UQAM-UO UCLP environment, a customer can use a graphical interface where available network resources are shown to the user, who can operate on them to create the desired connection.

Once the link is finished, the sequence of operations required to create it from the initial resources can be saved as a script and "played back" at a later time at the request of the user to provide him with the desired connection. Under the hood of this graphical engine is a web service environment. Each provider gives access to its resources in an Articulated Private Network (APN) via an LPO-factory web service from which LPOs can be controlled and consumed. Each LPO is identified by a unique ID, and the UCLP operations usually manipulate these IDs.

The script built by the user in the graphical interface is actually a BPEL process that invokes LPO operations by means of XML messages like for any other web service interaction. The corresponding BPEL operation for LPO concatenation takes as input an array of LPOs to concatenate. A simplified version of the concatenateRequest message structure is shown below:

<message>
  <operation>concatenateRequest</operation>
  <LPO-ID>i_1</LPO-ID>
  <LPO-ID>i_2</LPO-ID>
  ...
  <LPO-ID>i_n</LPO-ID>
</message>

The response of such an operation is the following:

<message>
  <operation>concatenateResponse</operation>
  <LPO-ID>i</LPO-ID>
</message>

Similarly, the corresponding BPEL operation for LPO partition takes as input the reference to an LPO and returns an array of references to spawned lightpaths, each of the desired bandwidth. A request is therefore of the following form:

<message>
  <operation>partitionRequest</operation>
  <LPO-ID>i</LPO-ID>
  <bandwidth>b</bandwidth>
  <login>ℓ</login>
</message>

where i is the ID of the LPO to partition, b is the bandwidth of the desired fragments and ℓ is a string representing the login name for accessing that resource. The response from this request is a message of the following form:

<message>
  <operation>partitionResponse</operation>
  <LPO-ID>i_1</LPO-ID>
  <LPO-ID>i_2</LPO-ID>
  ...
  <LPO-ID>i_n</LPO-ID>
</message>

Therefore, the GUI is just a lightpath-oriented rendition of a standard BPEL workflow design environment. Although the interface is adapted for LPO manipulation, the processes contain full-fledged BPEL code that can have loops, conditional branching, and even interact with other, non-UCLP web services. Moreover, users can bypass the GUI and program their own scripts involving UCLP resources using the BPEL environment of their choice. Figure 4 presents a simple pattern of messages exchanged between a customer BPEL process and a provider LPO-Factory service.

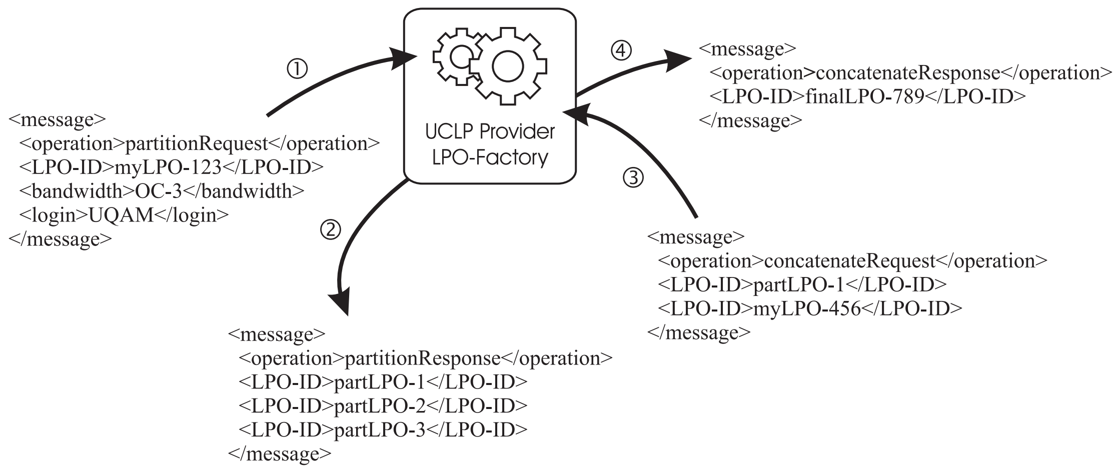
3.2 UCLP Service Constraints

While a service oriented network offers much more flexible use of network resources, a major bottleneck is making sure that these resources are correctly used. The consumer of lightpath resources from a provider is subject to two kinds of restrictions on its use and composition of LPOs:

- Technical constraints: these constraints arise because of the physical or logical nature of the resources involved in the operations.
- Policy constraints: these constraints arise for non-technical reasons, often dealing with business logic; this may include membership restrictions, QoS requirements or other reasons.

These constraints are capital to avoid publicizing erroneous services that could modify in a wrong way the physical resources they represent. We now proceed to show that a number of these constraints are data-aware temporal properties.

Let us examine the case of partition as a first example. This operation takes as input the ID of some LPO x and returns new LPOs y, z corresponding to the results of the partition. From there on, it does not make sense to again use x as an argument of a UCLP operation such as concatenate. Although the LPO still physically exists, it has been logically superseded by its fragments y, z. The script could even have applied further operations on y and z, like concatenating them to other LPOs or further partitioning them. In this context, invoking an operation with x is at best semantically unsound and at worst plain dangerous for the reliability of the whole UCLP environment. We must therefore enforce the following constraint on any UCLP script:



```
<message>
  <operation>partitionRequest</operation>
  <LPO-ID>myLPO-123</LPO-ID>
  <bandwidth>OC-3</bandwidth>
  <login>UQAM</login>
</message>
```

```
<message>
  <operation>partitionResponse</operation>
  <LPO-ID>partLPO-1</LPO-ID>
  <LPO-ID>partLPO-2</LPO-ID>
  <LPO-ID>partLPO-3</LPO-ID>
</message>
```

```
<message>
  <operation>concatenateRequest</operation>
  <LPO-ID>partLPO-1</LPO-ID>
  <LPO-ID>myLPO-456</LPO-ID>
</message>
```

```
<message>
  <operation>concatenateResponse</operation>
  <LPO-ID>finalLPO-789</LPO-ID>
</message>
```

Figure 4. Pattern of messages exchanged between a customer BPEL process and a provider LPO-Factory service

UCLP Service Constraint 1. *An LPO used as the input of an operation (concatenate, partition) cannot be the input of some other operation during the lifetime of the result of the first operation.*

This constraint is data-aware, as it relates the content of two messages at two different moments in time: the LPO IDs appearing in a request must be different from the LPO IDs appearing in any future request. A second constraint involves the concatenate operation. As has been said earlier, concatenated LPOs must have identical bandwidths. This can be formulated in the following way:

UCLP Service Constraint 2. *Every LPO occurring as an input of the concatenate operation must be of the same bandwidth.*

Constraints can also link together invocations of different operations. For example, the semantics of the concatenate operation supposes that the LPOs to be concatenated are adjacent (i.e. they have exactly one extremity in common). Therefore, although it would be semantically perfectly valid, it does not make sense to take two LPOs originating from the same partition operation and attempt to concatenate them, as these two LPOs are actually the same end-to-end connection. This calls for a third, mixed constraint:

UCLP Service Constraint 3. *If two LPOs are the result of the same partition response, they cannot be involved together in the same concatenate request.*

These first three constraints are in the realm of technical restrictions: they arise because of the specific nature of the web services involved.

Business logic can also be a source of data-aware temporal properties. Suppose a small UCLP resource provider wants to limit management overhead of its LPOs; it might want to expect from all users of its resources to avoid over-partitioning its links by imposing that LPOs can only be partitioned once.

UCLP Service Constraint 4. *If an LPO is the result of a partition response, it cannot be involved in a partition request.*

This constraint clearly has nothing to do with the semantics of the partition operation, but rather with some additional business logic imposed by one particular service provider.

Another constraint related to business logic is the following. Once an LPO is partitioned, the IDs for the spawned LPOs consisting of the fragments of the original LPO are returned to the user of the script. Because of UCLP Service Constraint 1, the original LPO cannot be used by anyone during the lifetime of its fragments. Therefore, partitioning an LPO without using any of its fragments in a future operation in a script wastes resources by making the original LPO unavailable to anybody without ever using it. A UCLP resource provider might therefore impose the following constraint:

UCLP Service Constraint 5. *At least one of the LPOs contained in a partition response must be involved in an operation at some point later in the script.*

With this solution, each UCLP service provider can define for its own services custom rules that take into account the specifics of each service: intra- or inter-domain, brand

and framing type (Ethernet, SONET), etc. These rules act both as restrictions that prevent a user from using a service incorrectly, and as guarantees that if respected, the service should behave as expected.

4 A Data-Aware Temporal Logic

The use of temporal logic to express the behaviour of a system is a common approach to most related work on verification of web service workflows. Temporal logics are commonly used in model checking for describing behavioural properties of systems. However, classical temporal formalisms are propositional, and Section 2 has shown how these works are only partially appropriate to the modelling and validation of data-aware properties. In this section, we introduce CTL-FO$^+$, an extension of the classical temporal logic CTL, and analyze the complexity of its model checking algorithm.

4.1 Workflow Modelling

We start by briefly recalling the basics of temporal logic. A transition system —also called a Kripke structure— is a set of Boolean variables and a directed graph where each node represents a state of the system uniquely identified by the values of the Boolean variables in this state. Formally, a Kripke structure is a quadruple $K = (S, I, R, L)$, where S is the set of states, I is the set of initial states, $R \subseteq S \times S$ is a transition relation and L is a labelling function $S \to 2^{AP}$, for AP a set of atomic propositions. For the sake of clarity, we can abuse notation and use state variables that take their values in arbitrary discrete sets instead of Boolean; the set AP is therefore the set of atomic propositions that encode into Boolean variables the discrete values occurring in the structure. Moreover, we can assume each structure has only one initial state by adding a dummy start state in the case it does not. The set of states, together with the transition relation, forms a directed graph. A path in this graph from a given start state is called an *execution sequence*.

In the present context, a suitable transition system for representing a web service workflow should model the actual messages that are exchanged. Each state represents a message that is either sent or received; it contains state variables that represent the content of that message. Such a transition system is a generalized construction of a classical *Moore machine* [18]. Any path in the system corresponds to a possible sequence of messages in a service interaction. Properties about message sequences become properties on sequence of states that can then be expressed using temporal logics.

4.2 Syntax and Semantics of CTL-FO$^+$

The Computation Tree Logic with Full First-order Quantification (CTL-FO$^+$) is an extension of the well-known temporal logic CTL [8]. CTL and a related logic called LTL are the most commonly used languages for describing sequentialities in finite-state systems. All major model checking tools, such as SPIN [17] and NuSMV [7], verify temporal formulæ expressed in one of these logics. The reader is referred to [8] for a deeper coverage of CTL and other temporal logics.

CTL-FO$^+$ is aimed at describing sequentialities in a finite-state system while allowing full quantification over data. Formulæ are built from Boolean variables and the constants *true* and *false* using the classical connectors: $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies) and $\neg$ (not). CTL-FO$^+$ further provides *temporal operators* that can be used on top of traditional propositional logic formulæ to specify the temporal conditions.

We briefly recall these operators, which are taken directly from CTL. They can be divided into two classes. The first class is composed of *universal* operators that assert properties about all executions starting from the current state. The first of these operators is **AG**, which means "globally". For example, the formula $\mathbf{AG}\,\varphi$ means that formula φ is true in every state of every execution starting at the current state. The operator **AF** means "eventually"; the formula $\mathbf{AF}\,\varphi$ is true whenever for all executions, φ holds for some future state. The operator **AX** means "next"; it is true whenever φ holds in any possible next state of the current state. Finally, the **AU** operator means "until"; the formula $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$ is true if, in any execution sequence, φ holds for all states until ψ holds.

The second class of operators are called *existential* and are designated by **EG**, **EF**, **EX** and **EU**; they are defined in the same way as their universal equivalents, except that the condition holds only for some instead of all possible sequences.

We extend the expressiveness of the traditional CTL by adding first-order quantification over state variables. The resulting language has the following formal syntax and semantics.

Definition 1 (Syntax)**.** *The language CTL-FO$^+$ (Computation Tree Logic with Full First-order Quantification) is obtained by closing CTL under the following construction rules:*

1. *If x is a variable or a constant, and y is either a variable, a constant or a state variable, then $x = y$ is a CTL-FO$^+$ formula;*

2. *If φ and ψ are CTL-FO$^+$ formulæ, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, $\mathbf{AG}\,\varphi$, $\mathbf{EG}\,\varphi$, $\mathbf{AF}\,\varphi$, $\mathbf{EF}\,\varphi$, $\mathbf{AX}\,\varphi$, $\mathbf{EX}\,\varphi$, $\mathbf{A}\,\varphi\,\mathbf{U}\,\psi$, $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$, are CTL-FO$^+$ formulæ;*

3. *If φ is a CTL-FO$^+$ formula and x is a free variable in φ, then $\exists x : \varphi(x)$ and $\forall x : \varphi(x)$ are CTL-FO$^+$ formulæ.*

Definition 2 (Semantics)**.** *Let $K = (S, I, R, L)$ be a transition system and $s_0 \in S$ be a state. Define a path $\pi = s_0, s_1, \ldots$ as a sequence of states in S such that $(s_i, s_{i+1}) \in R$ for every $i \geq 0$. Let Dom(x) be the (finite) domain of a quantified variable x, p be some state variable in K and c_1 and c_2 be constants. We say the pair K, s_0 satisfies the CTL-FO$^+$ formula φ if and only if it respects the following rules:*

$$\begin{array}{rcl}
K, s_0 \models p = c_1 & \Leftrightarrow & p \textit{ is equal to } c_1 \textit{ in state } s_0 \\
K, s_0 \models c_1 = c_2 & \Leftrightarrow & c_1 \textit{ is equal to } c_2 \\
K, s_0 \models \neg\varphi & \Leftrightarrow & K, s_0 \not\models \varphi \\
K, s_0 \models \varphi \vee \psi & \Leftrightarrow & K, s_0 \models \varphi \textit{ or } K, s_0 \models \psi \\
K, s_0 \models \mathbf{AF}\,\varphi & \Leftrightarrow & \textit{for each } \pi = s_0 s_1 s_2 \ldots, \\
 & & K, s_i \models \varphi \textit{ for some } i \\
K, s_0 \models \mathbf{EX}\,\varphi & \Leftrightarrow & \textit{there exists } \pi = s_0 s_1 s_2 \ldots \\
 & & \textit{such that } K, s_1 \models \varphi \\
K, s_0 \models \mathbf{E}\,\varphi\,\mathbf{U}\,\psi & \Leftrightarrow & \textit{there exists } \pi = s_0 s_1 s_2 \ldots \textit{ such} \\
 & & \textit{that } K, s_j \models \psi \textit{ for some } j \textit{ and} \\
 & & K, s_i \models \varphi \textit{ for } i < j \\
K, s_0 \models \exists x : \varphi(x) & \Leftrightarrow & \textit{there exists } a \in \mathit{Dom}(x) \textit{ such} \\
 & & \textit{that } K, s_0 \models \varphi(a)
\end{array}$$

By extension, we write $K \models \varphi$ if the initial state s_0 of K is such that $K, s_0 \models \varphi$.

The set of operators **AF**, **EX**, **EU**, $\neg$, $\vee$ and $\exists$ is called an *adequate* set of connectives in that all other operators can be derived from a combination of them. This result is classical [8].

CTL-FO$^+$ is reminiscent of [22] which introduces a logic called EQCTL that extends CTL by allowing existential quantification over *state variables*. EQCTL is not closed under negation; therefore, universal quantification cannot be obtained. CTL-FO$^+$ quantifies over *values* and is is closer to true first-order quantification, but is also richer: the model checking of EQCTL is NP-complete, while we show later that model checking in CTL-FO$^+$ is in a higher complexity class. A closer work is QCTL [28] which extends CTL by including first-order quantification and monadic second-order quantification over arbitrary *algebraic data structures*. Such expressiveness is not required in our case. Finally, CTL-FO$^+$ can freely mix temporal and data quantification without restriction. This is an extension over the logic CTL-FO defined in [9], which does not allow formulæ containing temporal operators to be existentially quantified. We suspect the inclusion to be strict: for example, it is not known whether CTL-FO can express UCLP Constraint 5, while we shall see later that CTL-FO$^+$ does. The following theorem is a natural consequence of this observation.

Theorem 1. *CTL-FO is a subset of CTL-FO$^+$.*

4.3 Formalizing Web Service Properties

The values of the variables appearing in a CTL-FO$^+$ formula are quantified according to specific parts of the XML message that is received or sent in the current state of the system. When referring to message data, it is never necessary to quantify over all values of all elements in the message; rather, we normally want to quantify for all values of a specific element name. To indicate this, we add a subscript to the quantifier indicating the name of the element. A quantifier like $\forall_{\text{LPO-ID}} x$ therefore means "for all values x of elements named LPO-ID in the current message".

Hence, UCLP Service Constraint 1 becomes the following CTL-FO$^+$ formula:

UCLP Formal Service Constraint 1.

$$\begin{array}{l}
\mathbf{AG}\,(\forall_{\mathit{operation}}\, x_1 : x_1 = \mathit{concatenateRequest} \rightarrow \\
\quad \forall_{\mathit{LPO\text{-}ID}}\, x_2 : \mathbf{AX}\,\mathbf{AG}\,(\forall_{\mathit{operation}}\, x_3 : \\
\quad\quad (x_3 = \mathit{partitionRequest} \vee x_3 = \mathit{concatenateRequest}) \\
\quad\quad\quad \rightarrow \forall_{\mathit{LPO\text{-}ID}}\, x_4 : x_2 \neq x_4))
\end{array}$$

This formula states that at any time in any execution of the script, if the operation x_1 of the message is concatenateRequest, then for every LPO ID x_2 appearing in this message, we have that for every future message whose operation element value x_3 is partitionRequest or concatenateRequest, any value x_4 for its LPO ID is different from x_2. In other words, once an LPO has been concatenated, no further partition or concatenation involves this LPO, which is indeed equivalent to UCLP Service Constraint 1. A similar formula constrains the use of the results from a partitionRequest.

In the same way, UCLP Service Constraint 2 can be enforced in various ways; in the present context, we can limit ourselves to assuming that all original LPOs have the same bandwidth, and check for example that no attempt is made to concatenate an LPO resulting from a partition operation with an LPO that has not been partitioned. We obtain the the following CTL-FO$^+$ formula:

UCLP Formal Service Constraint 2.

$$\begin{array}{l}
\mathbf{AG}\,(\forall_{\mathit{operation}}\, x_1 : x_1 = \mathit{partitionResponse} \rightarrow \\
\quad \forall_{\mathit{LPO\text{-}ID}}\, x_2 : \mathbf{AX}\,\mathbf{A}\,(\neg(\forall_{\mathit{operation}}\, x_3 : \\
\quad\quad \forall_{\mathit{LPO\text{-}ID}}\, x_4\, \forall_{\mathit{LPO\text{-}ID}}\, x_5 : \\
\quad\quad\quad x_3 = \mathit{concatenateRequest} \wedge x_2 = x_4) \\
\quad\quad\quad \mathbf{U}\,(\forall_{\mathit{operation}}\, x_6\, \forall_{\mathit{LPO\text{-}ID}}\, x_7 : \\
\quad\quad\quad\quad x_4 = \mathit{partitionResponse} \wedge x_5 = x_7)))
\end{array}$$

This formula states that at any time in any execution of the script, if a message received is a partition response, then for every LPO ID x_2 appearing in this message, no message with operation concatenateRequest involving x_4, when $x_2 = x_4$, can appear unless all other LPOs x_5 also result from a *previous* partitionResponse. In other words, a partitioned LPO can only be concatenated with other partitioned LPOs, which is equivalent to UCLP Service Constraint 2.

A simpler constraint is UCLP Service Constraint 3, which becomes in CTL-FO$^+$:

UCLP Formal Service Constraint 3.

$$\begin{array}{l}\mathbf{AG}\,(\forall_{\textit{operation}}\, x_1 : x_1 = \textit{partitionResponse} \rightarrow \\ \quad \forall_{\textit{LPO-ID}}\, x_2\, \forall_{\textit{LPO-ID}}\, x_3 : \mathbf{AX}\,\mathbf{AG} \\ \quad (\forall_{\textit{operation}}\, x_4\, \forall_{\textit{LPO-ID}}\, x_5\, \forall_{\textit{LPO-ID}}\, x_6 \\ \quad\; x_4 = \textit{concatenateRequest} \rightarrow \\ \quad\quad (x_2 \neq x_5 \wedge x_3 \neq x_6)))\end{array}$$

UCLP Service Constraint 4 has exactly the same structure as UCLP Service Constraint 1. We will omit it from the remainder of this paper; the reader can assume that any technical discussion on Service Constraint 1 also applies to Service Constraint 4.

Finally, UCLP Service Constraint 5 is the following:

UCLP Formal Service Constraint 5.

$$\begin{array}{l}\mathbf{AG}\,(\forall_{\textit{operation}}\, x_1 : x_1 = \textit{partitionResponse} \rightarrow \\ \quad \exists_{\textit{LPO-ID}}\, x_2 : \mathbf{AF}\,(\forall_{\textit{operation}}\, x_3 : \\ \quad\; x_3 = \textit{concatenateRequest} \wedge \\ \quad\quad \forall_{\textit{LPO-ID}}\, x_4 : x_2 = x_4))\end{array}$$

5 Validating CTL-FO$^+$ Properties

In this section, we show how CTL-FO$^+$ formulæ can be actually validated on a web service workflow by presenting a complete model checking algorithm. The complexity of this algorithm is then established and discussed. In particular, we show that CTL-FO$^+$ model checking is a problem as tractable as the LTL model checking problem that is widely used in the industry, and that any web service model that uses a data-aware workflow, but *propositional* properties cannot efficiently simulate data-awareness.

5.1 Model Checking CTL-FO$^+$

We now present a complete algorithm for model checking CTL-FO$^+$ formulæ. It is derived from the classical CTL model checking algorithm and is presented below. The procedure CHECK performs by structural induction on the CTL-FO$^+$ formula φ and consists in forming recursively the set of states s such that $K, s \models \varphi$. If the subformula to check is of the form $\neg\varphi$, $\varphi \wedge \psi$, $\mathbf{AF}\,\varphi$, $\mathbf{EX}\,\varphi$, $\mathbf{E}\,\varphi\,\mathbf{U}\,\psi$, the algorithm is identical to the model checking of a CTL formula.

Differences arise when the main operator of the formula is an existential quantifier, $\exists x : \varphi(x)$. In such a case, the algorithm successively applies the model checking of $\varphi(x)$ for each value a in the domain of x and keeps states which are in at least one of the subsets. Finally, model checking of ground terms is composed of equality testing. Two cases must be considered. For an equality the form $c_1 = c_2$, where c_1 and c_2 are constants, either the entire Kripke structure satisfies it if the assertion is true, or no state satisfies it if the assertion is false. For an equality of the form $p = c$, where p is a state variable, the set of states where $p = c$ is returned.

A Kripke structure K satisfies the global CTL-FO$^+$ formula φ if and only if its initial state is in the set returned by CHECK(φ).

5.2 CTL-FO$^+$ Model Checking is Tractable

We now establish the complexity of model checking CTL-FO$^+$ formulæ and show that data-aware properties cannot be modelled effectively by propositional properties.

Theorem 2. *Let φ be a CTL-FO$^+$ formula and K be a Kripke structure over finite domains. Determining whether $K \models \varphi$ is decidable and is PSPACE-complete.*

Proof. We first show the model checking problem is PSPACE-hard by reducing the quantified Boolean formula problem (QBF), known to be PSPACE-complete [15], to CTL-FO$^+$ model checking. A quantified Boolean formula φ is of the form $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi$, where Q_1 is either the existential ($\exists$) or the universal ($\forall$) quantifier and the x_i are Boolean variables (their domain is $\{0, 1\}$). In fact, φ is a CTL-FO$^+$ formula with no temporal operators and which refers to no state variables. Therefore, if φ is satisfiable, then $K \models \varphi$ for every Kripke structure K; conversely, if φ is unsatisfiable, then $K \not\models \varphi$ for every K. Therefore, QBF satisfiability can be solved using CTL-FO$^+$ model checking of an arbitrary Kripke structure.

The second step consists in showing that the procedure CHECK is in PSPACE. It suffices to observe that each recursive call returns a subset of the set of states of the Kripke structure to check; therefore, the space consumed by each recursive call is constant. Since the number of calls is bounded by the depth of the formula, this algorithm is polynomial in the size of the CTL-FO$^+$ formula and the transition system. Remark that the PSPACE class of decision problems only requires polynomial use of *memory space*; the algorithm is clearly exponential with respect to time. □

The PSPACE-completeness result places CTL-FO$^+$ model checking for finite domains in the same complexity

```
Procedure CHECK(AF $\varphi$)
  $M := $ CHECK($\varphi$)
  Do
    $N := \emptyset$
    For each $s_1 \in S$
      $flag := true$
      For each $s_2 \in S$
        If $(s_1, s_2) \in R$ and $s_2 \notin M$ then
          $flag := false$
        End if
      End for
      If $flag = true$ then
        $N := N \cup \{s_1\}$
      End if
    End for
    $M := M \cup N$
  Loop until $N = \emptyset$
  Return $M$
End procedure

Procedure CHECK($\neg\varphi$)
  Return $S \setminus$ CHECK($\varphi$)
End procedure

Procedure CHECK($\varphi \vee \psi$)
  Return CHECK($\varphi$) $\cup$ CHECK($\psi$)
End procedure

Procedure CHECK(E $\varphi$ U $\psi$)
  $M := $ CHECK($\psi$)
  Do
    $N := \emptyset$
    For each $(s_1, s_2) \in R$
      If $(s_1, s_2) \in R$ and $s_2 \in M$ then
        $N := N \cup \{s_1\}$
      End if
    End for
    $M := M \cup N$
  Loop until $N = \emptyset$
  Return $M$
End procedure

Procedure CHECK(EX $\varphi$)
  $M := $ CHECK($\varphi$)
  $N := \emptyset$
  For each $(s_1, s_2) \in R$
    If $s_2 \in N$ then
      $N := N \cup \{s_1\}$
    End if
  End for
  Return $N$
End procedure

Procedure CHECK($\exists x : \varphi(x)$)
  $N := \emptyset$
  For each $a \in Dom(x_i)$ do
    $N := N \cup$ CHECK($\varphi(a)$)
  End for
  Return $N$
End procedure

Procedure CHECK($p = c$)
  $N := \emptyset$
  For each $s \in S$ do
    If $p = c$ in state $s$ then
      $N := N \cup \{s\}$
    End if
  Return $N$
End procedure

Procedure CHECK($c_1 = c_2$)
  If $c_1 = c_2$ then
    Return $S$
  Else
    Return $\emptyset$
  End if
End procedure
```

Table 1. The recursive model checking procedure for CTL-FO$^+$

class as model checking of an LTL formula [8]. LTL is a temporal logic widely used in the industry in conjunction with the SPIN model checker, and many works with *propositional* properties mentioned in section 2.2 use LTL as their language for expressing constraints on message sequences. Therefore, although CTL-FO$^+$ allows to fully access the data content of the messages, its model checking problem is no less tractable than many other existing solutions that do not provide data-aware temporal capabilities.

5.3 Simulating Data-awareness with Propositional Properties

Studying the complexity of the CTL-FO$^+$ model checking algorithm can teach us more. Since the domains for each variable are considered finite, it is possible to use the semantics of Definition 2 and convert each quantifier into a conjunction or a disjunction of a finite number of terms. The resulting expression is a plain CTL formula where all references to data are static. In turn, the expansion of a CTL-FO$^+$ formula φ into a propositional CTL formula φ' is exponential in the number of quantifiers, since each quantified subformula must be repeated once for each possible value in the domain. However, the model checking algorithm of a CTL formula in in P: it has a worst-case running time linear in the size of the formula to check, and linear in the size of the Kripke structure. Therefore, using CTL model checking on φ' takes exponential time, which is no worse than using CTL-FO$^+$ model checking on φ.

One might then think that CTL-FO$^+$ is simply CTL with an additional level of syntactic sugar, and that data-aware workflows with *propositional* properties, as described in Section 2.2, are already sufficient to model any data-aware property by simply extending the message alphabet. However, this is not the case; the following theorem shows that a translation of CTL-FO$^+$ to CTL is highly unlikely to lead to an algorithm more efficient than the procedure CHECK described previously.

Theorem 3. *If there exists a polynomial reduction of CTL-FO$^+$ model checking to CTL model checking, then P = NP.*

Proof. A polynomial reduction of CTL-FO$^+$ model checking to CTL model checking entails that for every Kripke structure K and every CTL-FO$^+$ formula φ, there exists a Kripke structure K' and a CTL formula φ' such that $K \models \varphi$ if and only if $K' \models \varphi'$. Moreover, the size of K' and φ are respectively polynomial in the sizes of K and φ. Since CTL-FO$^+$ model checking is PSPACE-complete and CTL model checking is in P, we have PSPACE $\subseteq$ P. The result follows since P $\subseteq$ NP $\subseteq$ PSPACE. $\square$

Therefore, unless P $=$ NP, any attempt at using data-aware workflows with propositional properties to model data-aware properties will either blow the size of the formulæ or the size of the model by an exponential factor; the PSPACE-completeness of the model checking algorithm is not preserved and therefore the translation is not optimal. In other words, CTL-FO$^+$ is exponentially more succinct and efficient than any propositional modelling of data-awareness.

More practical reasons also justify this position. The translation of constraints into temporal logic becomes tightly coupled with the actual script on which it has to be checked. This is because the translation of the quantifiers shown depends on the values occurring in the script. It is, however, unrealistic that a UCLP resource provider advertises its constraints in such a manner: one would have to know in advance all possible LPO values occurring in scripts prepared by third-parties to include them in the large disjunction.

6 Conclusion

In this paper, we have shown how “data-aware” temporal properties can be used to express constraints on the behaviour of a web service composition. These properties enable complex temporal relationships to be expressed, while at the same time allowing full first-order quantification on the content of the messages. We presented a real-world scenario where data-aware properties arise naturally, and showed how existing related work is only partially appropriate for the validation of such properties. To this end, we introduced the logic CTL-FO$^+$, showed its model checking algorithm and studied its complexity. We conclude that model checking data-aware temporal properties is a tractable problem and that any web service model that uses a data-aware workflow, but *propositional* properties cannot efficiently simulate data-awareness.

This project lends itself to many further developments. We are currently working on an adaptation of the NuSMV model checker to CTL-FO$^+$ and on efficient CTL reduction techniques. Moreover, static, *a priori* model checking on scripts could be complemented with runtime monitoring of CTL-FO$^+$ properties for a given transaction.

References

[1] *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong*. IEEE Computer Society, 2006.

[2] J. Arias-Fisteus, L. S. Fernández, and C. D. Kloos. Applying model checking to BPEL4WS business collaborations. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 826–830. ACM, 2005.

[3] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In Böhm et al. [4], pages 613–624.

[4] K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P.-Å. Larson, and B. C. Ooi, editors. *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM, 2005.

[5] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105:73–94, 2004.

[6] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.

[7] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An opensource tool for symbolic model checking. In E. Brinksma and K. G. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 2000.

[9] A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web services. In A. Deutsch, editor, *PODS*, pages 71–82. ACM, 2004.

[10] A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In S. Vansummeren, editor, *PODS*, pages 90–99. ACM, 2006.

[11] Z. Duan, A. J. Bernstein, P. M. Lewis, and S. Lu. A model for abstract process specification, verification and composition. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *ICSOC*, pages 232–241. ACM, 2004.

[12] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based analysis of obligations in web service choreography. In *AICT/ICIW*, page 149. IEEE Computer Society, 2006.

[13] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW*, pages 621–630. ACM, 2004.

[14] X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. In G. S. Avrunin and G. Rothermel, editors, *ISSTA*, pages 252–262. ACM, 2004.

[15] M. R. Garey and D. S. Johnson. *Computers and intractability, a guide to the theory of NP-completeness*. W. H. Freeman, 1979.

[16] P. Greenfield, A. Fekete, J. Jang, and D. Kuo. Compensation is not enough. In *EDOC*, pages 232–239. IEEE Computer Society, 2003.

[17] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[18] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation, Second Edition*. Addison Wesley, 2000.

[19] J. E. Johnson, D. E. Langworthy, L. Lamport, and F. H. Vogt. Formal specification of a web services protocol. *Electr. Notes Theor. Comput. Sci.*, 105:147–158, 2004.

[20] R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of communication models in web service compositions. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 267–276. ACM, 2006.

[21] M. Koshkina and F. van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *ACM SIGSOFT SEN*, 29(5), September 2004.

[22] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In P. Wolper, editor, *CAV*, volume 939 of *Lecture Notes in Computer Science*, pages 325–338. Springer, 1995.

[23] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing interacting BPEL processes. In S. Dustdar, J. L. Fiadeiro, and A. P. Sheth, editors, *BPM*, volume 4102 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2006.

[24] G. Meredith and S. Bjorg. Contracts and types. *Commun. ACM*, 46(10):41–47, 2003.

[25] S. Nakajima. Model-checking of safety and security aspects in web service flows. In N. Koch, P. Fraternali, and M. Wirsing, editors, *ICWE*, volume 3140 of *Lecture Notes in Computer Science*, pages 488–501. Springer, 2004.

[26] S. Parastatidis, J. Webber, S. Woodman, D. Kuo, and P. Greenfield. SOAP service description language (SSDL). Technical Report CS-TR-899, University of Newcastle, Newcastle upon Tyne, 2005.

[27] M. Pistore, M. Roveri, and P. Busetta. Requirements-driven verification of web services. *Electr. Notes Theor. Comput. Sci.*, 105:95–108, 2004.

[28] A. Rensink. Model checking quantified computation tree logic. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2006.

[29] K. Schmidt and C. Stahl. A Petri net semantic for BPEL4WS validation and application. In *Proceedings of the 11th Workshop on Algorithms and Tools for Petri Nets (AWPN 04) / Ekkart Kindler (Ed.)*, pages 1–6. Bericht tr-ri-04-251, Universität Paderborn, September 2004.

[30] K. J. Turner. Formalising web services. In F. Wang, editor, *FORTE*, volume 3731 of *Lecture Notes in Computer Science*, pages 473–488. Springer, 2005.

[31] M. Venzke. Specifications using XQuery expressions on traces. *Electr. Notes Theor. Comput. Sci.*, 105:109–118, 2004.

[32] C. D. Walton. Model checking multi-agent web services, 2004.

[33] J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, and G. Decker. Service interaction modeling: Bridging global and local views. In *EDOC* [1], pages 45–55.