The Journal of Systems and Software 83 (2010) 1651–1661

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A model checker for WS-CDL

Hongbing Wang [a,*], Zuling Kang [a], Ning Zhou [a], Li Li [b]

[a] School of Computer Science and Engineering, Southeast University, Nanjing, PR China
[b] Faculty of Computers and Information Science, Southwest University, Chongqing 400715, PR China

ARTICLE INFO

Article history:
Received 15 March 2009
Received in revised form
24 December 2009
Accepted 30 March 2010
Available online 15 June 2010

Keywords:
Web service composition
TLA
WS-CDL
TLA4CDL
Model checker

ABSTRACT

Service computing is becoming the prominent paradigm for distributed computing and electronic businesses. It enables developers to rapidly create their own software to meet the demands of their business processes, by composing existing services, especially Web services distributed on the Internet. WS-CDL is a W3C-proposed language for Web service composition, featuring the peer description of composite Web services amongst multiple participants. Since the traditional model checking methods based on the linear temporal logic (LTL) has limit in expressing the state-action relationship for a composite Web service model, this paper proposes a new approach, based upon the idea of Temporal Logic of Actions (TLA), to model check the composite Web services described in WS-CDL. In this paper, WS-CDL is extended by a new sub-language for expressing the temporal and action restriction properties, named TLA4CDL. The expressiveness of TLA4CDL is also discussed. The optimizing method called partial order reduction is introduced, followed by the discussion of the model checker algorithm. This leads to the development and implementation of the WS-CDL model checker. Finally, several test scenarios are provided in order to validate the WS-CDL model checker. The experimental results demonstrate this model checker is capable of detecting deadlock and verifying the specified constraint attributes by TLA4CDL. A comparison of experimental results with and without the partial order reduction method shows that our checker has better performance.

1. Introduction

The promise of reusability in software systems has become a common theme in commercial application development in recent years. It is elaborated with the development of Web services and the generic concept of Service-Oriented Architecture (SOA). Web services are autonomous software systems identified by URIs, which can be advertised, located, and accessed through messages encoded according to XML-based standards, and transmitted using Internet protocols. They are designed to support interoperable machine-to-machine interaction over a network, and are now becoming the main building blocks of the SOA. The SOA is a computing paradigm that utilizes services as fundamental elements for developing applications/solutions (Tan et al., 2009; Ardagna and Pernici, 2007; Oh et al., 2008; Lécué et al., 2008), with the help of service composition/collaboration technology such as WS-BPEL (OASIS, 2007) and WS-CDL (Kavantzas et al., 2005). Because Web services are based on the open and widely accepted protocols (e.g. HTTP and SOAP), they are capable of providing a uniform distributor for a wide range of computing devices and software platforms. Web services and SOA constitute the next major step in software engineering.

WS-CDL is a Web service collaboration specification for peer participants. It resides in the collaboration layer of the service protocol stack, and stands on a global viewpoint and describes the common and complementary observable behavior of all the participants involved (Kavantzas et al., 2005). The global nature of WS-CDL is capable of providing behaviors of all parties participating in a composition process within one single WS-CDL document, which presents extra convenience in some fields like modeling, reasoning and verification.

The intended users of WS-CDL include IT engineers and business analysts. In practice, organizations involved in a collaborative business process will negotiate on how this multiple participant related process should be carried out. Such negotiation usually includes the conditional and temporal constraints on some key properties and actions. When the negotiation is completed, the outcome can be drafted as Web service choreography in WS-CDL. Accordingly, each participant is able to create its own part of collaborative processes based on the description of the choreography.

A typical WS-CDL scenario might involve multiple parties, for example, a consumer, a retailer, a bank and a shipper. Initially, the consumer places an order by sending a message to the retailer. The

* Corresponding author.
E-mail addresses: hbw@seu.edu.cn (H. Wang), lily@swu.edu.cn (L. Li).

0164-1212/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2010.03.076

retailer confirms the ordering request and sends back the banking account number. To speed up the whole process, the retailer instructs the shipper to deliver the product while the retailer waits for payment confirmation from the bank. However, the product can only be handed to the consumer after the payment has been confirmed and received by the retailer.

The product delivery activity and payment activity can be executed in parallel in the above retailer-shipper scenario. While introducing the *parallel* structure into WS-CDL has a lot of advantages, it also makes the encoded WS-CDL choreography error-prone. In this scenario, a rule may be enforced that the product cannot be handed to the consumer *before* the payment confirmation arrives at the retailer.[1] Trivial bugs might have been introduced because of this. Some bugs are so trivial that they can be difficult to find manually. Instead, automatic methods such as formal verification are more favored.

Although formal verification of composite Web services constitutes an essential part in studying WS-CDL, the WS-CDL specification itself does not provide any facilities to support this functionality. Amongst the work in WS-CDL verification research, a commonly used method is to translate a WS-CDL document into the Promela model (Holzmann, 1991), describe the constraint attributes with the expression of LTL (Pnueli, 1981), and use SPIN (Holzmann, 1997) to model check (Vardi and Wolper, 1986; Courcoubetis et al., 1992) whether the model satisfies the expression (Zhao et al., 2006). However, there are some issues regarding this method.

The first issue is the expressiveness of LTL. LTL is a popular formal toolkit used in model checking, but it has some other issues. For instance, traditional LTL does not allow relating specifications to be written at different levels of abstraction since it is based on a global notion of 'next state' (Lamport, 1983). This limitation makes LTL inconvenient in dealing with relationships between states and actions in a software system. For example, we often need to express attributes such as 'continuously sending the connecting requests before the server acknowledges' or 'the server accepting or rejecting these connecting requests'. It is unlikely to be able to express these attributes with LTL alone.

One of solutions here is to employ the state pairs, such as $<s, s'>$, to represent the actions. Suppose there is an action $\alpha_1 = <s_1, s_2>$, then $\Box\alpha_1$ (the semantics of this LTL formula are defined as: globally α_1 has to hold on the entire subsequent path) can be represented as $\Box(s_1 \wedge \mathbf{X}s_2)$ (the temporal operator $\mathbf{X}s_2$ means that s_2 has to be hold in the next state). Initially, this trade-off seems feasible, but some problems are actually hidden in it. Consider another action α_2 to be used to increase a *state variable x* by one each time it is invoked. Suppose the domain of x is the integers below 1000, and s_i is used to denote $x = i$ in state s, then the LTL expression to indicate $\Box\alpha_2$ will be $\Box(s_1 \wedge \mathbf{X}s_2 \vee s_2 \wedge \mathbf{X}s_3 \vee \ldots \vee s_{999} \wedge \mathbf{X}s_{1000})$. It implies that the complexity of the LTL model checking algorithm is $2^{O(|CLOSURE(\varphi)|)} * O(|S|)$ (Clarke et al., 2000; Huth and Ryan, 2004) with φ the modal expressions and S the number of the state space. Obviously, such a complex expression will take a great amount of time to perform model checking. Note that some complex expressions used in practice cannot even be computed in a finite amount of time.

To cope with this problem, a TLA-based formal language to express the constraint attributes of a WS-CDL model is presented. Temporal logic of actions (Lamport, 1994, 2002), or TLA, is a formal language proposed by Lamport in the early 1990s. It aims at describing and verifying parallel systems. It combines the properties of temporal logic and the logic of actions. The grammar and the formal semantic of TLA is simple but without losing its expressive power feature. Unlike LTL, TLA is able to express the state-action relationship in parallel in the presence of actions. By using the attributes in TLA, $\Box\alpha_2$ in the previous example can be simply written as $\Box[x' = x + 1]_x$. Moreover, when the above action expression is model checked by TLC (Lamport, 2002) (the TLA model checker), unlike LTL, it will not lead to exponential computing time as LTL did.

```
<tla:predictDefinitions>
  <tla:action name="conformPayment" activity="ConformPayment" />
  <tla:action name="conformDelivery" activity="ConformDelivery " />
</tla:predictDefinitions>
<tla:restrictions>
  <tla:finite>
    <tla:sequential actionsRef="conformPayment conformDeliver" />
  </tla:finite>
</tla:restrictions>
```

Fig. 1. The TLA4CDL rule which enforces the rule in the retailer–shipper example that *ConformPayment* has to occur before *ConformDelivery*.

The second issue is the definition of actions. In TLA, actions are expressed by relationships of variables between two successive states. For example, when expressing the idea that variable x will always increase by 1, it says in TLA that the value of x in the next state is one greater than that of the current state and it is always true, or $\Box[x' = x + 1]_x$ in TLA formula. This action expressing approach works well in handling hardware model or low-level software model, since any action in a process will eventually change its variable values. However, when dealing with high-level software models like WS-CDL, it is not always the case.

To address this issue, let us reconsider the rule that must be enforced in the retailer–shipper scenario. Suppose there are two WS-CDL activities named *ConfirmPayment* and *ConfirmDelivery* respectively. Assume that in one occasion, *ConfirmPayment* sends the *tns:paymentConfirmation* message from the bank to the retailer while *ConfirmDelivery* sends the *tns:confirmDelivery* message from the retailer to the shipper. Since the value of the *tns:paymentConfirmation* variable is unpredictable, it cannot be expressed via TLA's method. The same is true for *tns:confirmDelivery*. To solve this problem, high-level actions are needed. With the characteristics of high-level actions in TLA4CDL, we believe it is able to solve the above problem.

TLA4CDL is a formal language proposed in this paper to express the temporal constraints of WS-CDL. As WS-CDL is closely related to Web services, an invocation to a Web service operation can then be defined as an action. Moreover, the basic activities and defined choreographies of WS-CDL can also be defined as actions. These actions are so called high-level actions defined in TLA4CDL.

With the retailer–shipper example, the enforced rule can be expressed in TLA4CDL (shown in Fig. 1). We will revisit it in Section 5 to demonstrate the verification of this TLA4CDL constraint.

The elements enclosed within *tla:predictDefinition* are used to define two actions, each of which refers to an activity used in WS-CDL. The functor *tla:sequential* is a TLA4CDL functor which means the actions listed in the *actionRef* attribute should be executed in a specified order, i.e. according to Fig. 1, the action *conformPayment* is required to execute before *conformDeliver*. The detailed syntax, semantics and expressiveness of TLA4CDL will be discussed in Section 3. It is worth noting that by introducing boolean composition (formally defined in Section 3.2) of seven basic functors, the complexity of the TLA4CDL-based model checking algorithm is successfully reduced compared with that of LTL.

The third issue is about TLA4CDL's trade-off between expressiveness and performance, in which performance refers to the complexity of the WS-CDL model checking algorithm. In TLA4CDL, seven basic TLA4CDL functors and their boolean composition are introduced to express the temporal constraints of a WS-CDL model. Although the boolean composition method makes TLA4CDL not as expressive as TLA, it is still good enough to express the commonly

[1] We will show how to express this rule via the TLA4CDL constraint later Section I. Briefly, TLA4CDL is a kind of TLA extension to WS-CDL.