Pattern Based Property Specification and Verification for Service Composition

Jian Yu[1, 2], Tan Phan Manh[1], Jun Han[1], Yan Jin[1], Yanbo Han[2], and Jianwu Wang[2]

[1] Faculty of ICT, Swinburne University of Technology, 3122 Hawthorn, Australia
{jyu, phan_tan, jhan, yjin}@ict.swin.edu.au
[2] Grid and Service Computing Research Center, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China 100080
{yujian, yhan, wjw}@software.ict.ac.cn

Abstract. Service composition is becoming the dominant paradigm for developing Web service applications. It is important to ensure that a service composition complies with the requirements for the application. A rigorous compliance checking approach usually needs the requirements being specified in property specification formalisms such as temporal logics, which are difficult for ordinary software practitioners to comprehend. In this paper, we propose a property pattern based specification language, named PROPOLS, and use it to verify BPEL service composition schemas. PROPOLS is easy to understand and use, yet is formally based. It builds on Dwyer et al.'s property pattern system and extends it with the logical composition of patterns to accommodate the specification of complex requirements. PROPOLS is encoded in an ontology language, OWL, to facilitate the sharing and reuse of domain knowledge. A Finite State Automata based framework for verifying BPEL schemas against PROPOLS properties is also discussed.

1 Introduction

Web service composition is emerging as a promising technology for the effective integration of applications across globally distributed organizations [1, 2]. When encapsulating modular business functions as standard Web services, cross-organizational business processes can be built with a service composition language like BPEL [3] or BPML [4].

It is important to ensure the behavioral compliance between a service composition application and the requirements. Unexpected application behaviors may not only lead to mission failure, but also may bring negative impact on all the participants of this process. Model checking [5] is a formal approach to software behavioral compliance checking. In this approach, a software application is abstracted as a formal model like Labeled Transition Systems (LTS), Finite State Automata (FSA), Petri nets, or process algebra. The behavioral requirements are specified as properties in formalisms such as Linear Temporal Logic (LTL), Computation Tree Logic (CTL), or Quantified Regular Expressions (QRE). Then the formal model can be verified against the specified requirements/properties through exhaustive state space

exploration. A serious problem, however, prevents the wide adoption of this approach. That is, the formal properties are surprisingly difficult to write for practitioners, who usually don't have solid mathematical backgrounds [6, 7].

In this paper, we present a lightweight specification language called PROPOLS (Property Specification Pattern Ontology Language for Service Composition), and an associated approach to the verification of BPEL schemas. PROPOLS is an OWL-based high-level pattern language for specifying the behavioral properties of service composition applications. PROPOLS is based on Dwyer et al.'s property patterns [6], which are high-level abstractions of frequently used temporal logic formulae. The property patterns enable people who are not experts in temporal logics to read and write formal specifications with ease and thus make model checking tools more accessible to common software practitioners [8]. Although it is claimed in [6] that patterns can be nested, no further work has been done on how to define composite patterns and what are their semantics. PROPOLS refines/extends the original pattern system in [6] by introducing the logical composition of patterns. This mechanism enables the definition of complex requirements in terms of property patterns, which is previously difficult or even impossible. PROPOLS uses the Web Ontology Language (OWL) as its base language. This makes PROPOLS properties sharable and reusable within/across application domains.

In addition to the PROPOLS language, we present a verification framework for checking the compliance of BPEL schemas against PROPOLS properties. The key techniques used include representing the semantics of PROPOLS properties as FSAs, representing the semantics of a BPEL schema as a LTS/FSA using Foster's BPEL2LTS tool [9], and checking the language inclusion between these two FSAs. The verification approach is illustrated using a non-trivial example.

This paper is structured as follows. In the next section, a motivating example scenario is presented. Section 3 gives a brief introduction to the original property specification patterns. Section 4 presents PROPOLS, including its syntax and semantics. Section 5 introduces the BPEL verification framework and illustrates the verification approach based on the example scenario. Finally, we discuss the related work in section 6 and conclude the paper in section 7.

2   A Motivating Scenario

Let's assume a computer manufacturing company AdvantWise is to provide an online purchasing service. The key requirements to this service are sketched as follows:

1) Customers can place purchase orders online. Any order should be checked before being processed. For an invalid order, the customer will get a rejection notification. For a valid one, the customer will get a confirmation.

2) The transactions between customers and AdvantWise follow a hard credit rule. That is, on the one hand, the customer pays to AdvantWise only when she has received the ordered products. On the other hand, AdvantWise processes the order only when it is confirmed that the customer will pay if the order is fulfilled. For this to be possible, a third party, bank, is introduced. To let AdvantWise start processing order, the customer must deposit the payment to the bank first. Then the bank will notify AdvantWise that the payment for the order has been already

kept in the bank. Anytime when the order is canceled, the payment will be refunded by the bank. If the order is fulfilled, the bank ensures that the payment is transferred to AdvantWise.

3) To fulfill an order, AdvantWise first checks its inventory. If there are enough products in the inventory, the products of the ordered quantity are packaged for shipping. If not, AdvantWise will initiate an emergency production schedule.

Clearly, the above-stated requirements express certain business constraints that the system implementation must follow. On the one hand, it is not easy to translate the embedded behavioral constraints into temporal logics in a straightforward manner. On the other hand, Dwyler's patterns are not expressive enough to state all the constraints, for example, the mutual exclusion between rejection and confirmation. After introducing PROPOLS in section 4, we will formulate all these requirements in PROPOLS, against which the BPEL service composition being designed in section 5 will be checked.

3   Property Specification Patterns

Property specification patterns were first proposed by Dwyer et al in [6]. These patterns include a set of commonly occurring high-level specification abstractions for formalisms like LTL, CTL or QRE. They enable people who are not experts in such formalisms to read and write formal specifications. According to [10], 92% of 555 property specifications collected from different sources matched one of the patterns.

A pattern property specification consists of a *pattern* and a *scope*. The pattern specifies *what* must occur and the scope specifies *when* the pattern must hold.

Patterns are classified into occurrence patterns and order patterns. We briefly describe the meaning of the important patterns below (the symbol P or Q represents a given state/event). Details of these patterns can be found in [11].

- **Absence:** P does not occur within a scope.
- **Universality:** P occurs throughout a scope.
- **Existence:** P must occur within a scope.
- **Bounded Existence:** P must occur at least/exactly/at most k times within a scope.
- **Precedence:** P must always be preceded by Q within a scope.
- **Response:** P must always be followed by Q within a scope.

A scope defines a starting and an ending state/event for a pattern. There are five basic kinds of scopes:

- **Globally:** the pattern must hold during the entire system execution.
- **Before:** the pattern must hold up to the first occurrence of a given P.
- **After:** the pattern must hold after the first occurrence of a given P.
- **Between…And:** the pattern must hold from an occurrence of a given P to an occurrence of a given Q.
- **After…Until:** the same as "between…and", but the pattern must hold even if Q never occurs.

The semantics of pattern properties may be given in LTL, CTL, QRE, or FSA [6, 7, 12, 13]. Fig. 1 illustrates the FSA semantics by three pattern properties. There, the

symbol 'O' denotes any other state/event than P and Q. Fig. 1(a) indicates that, before P occurs, an occurrence of Q is not accepted by the FSA. Fig. 1(b) states that if Q has occurred, an occurrence of P is necessary to drive the FSA to a final state. Finally, Fig. 1(c) says that only the occurrence of P can make the FSA reach a final state.
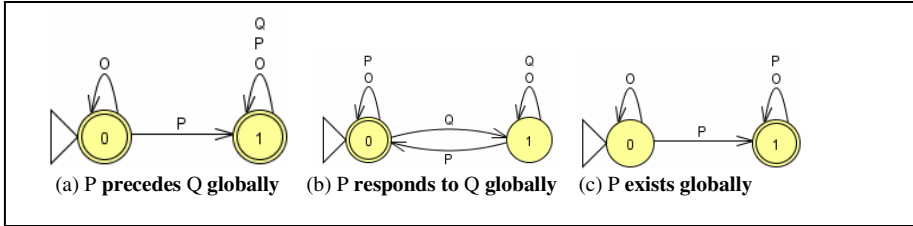

(a) P **precedes** Q **globally**   (b) P **responds to** Q **globally**   (c) P **exists globally**

Fig. 1. FSA Semantics of Example Pattern Properties

4   The PROPOLS Language

In this section, we first illustrate the structure and main components of the PROPOLS language. We then explain the syntax and semantics of the composite pattern, which are refinements/extensions to the original pattern system. A composite pattern is built by connecting pattern properties using logic operators. The main benefit of the composite pattern mechanism lies in that it enables the specification of complex requirements. Finally, the PROPOLS properties for the example scenario are given.

4.1   Language Structure

We designed PROPOLS as an ontology language for two purposes: First, ontology can be used to define standard terminology for the pattern system. Second, practitioners like business experts and system analysts can use concepts from existing domain ontologies to define pattern properties at a high level of abstraction. The benefits include: The pattern properties themselves are also become part of the shared formal domain knowledge, so they can be reused in a wide scope, and off-the-shelf semantic Web techniques can be used to map properties to related Web services automatically. At present, OWL is the most widely used Web ontology language. So we choose OWL as the base language of PROPOLS.

Fig. 2 shows a graphical overview of the PROPOLS ontology. It is generated from Ontoviz [14], an ontology visualization plug-in for an OWL editor tool Protégé. Its major elements are explained below.

OrderPattern: This class defines the set of order patterns we discussed in section 3. We use a more natural name *LeadsTo* to replace the original name *RespondsTo*. Here, P *LeadsTo* Q is the same as Q *RespondsTo* P.

OccurrencePattern: This class defines the set of occurrence patterns we discussed in section 3.

Scope: As discussed in section 3, every elementary pattern will be associated with a scope coming from *globally*, *before*, *after*, *between…and* or *after…until*.

Fig. 2. PROPOLS Ontology

Operation: This is the class for service operations. An operation is considered as an event in the pattern properties. It has a *provider* property linking to its provider and a *conceptReference* property linking to the corresponding concept in some ontology. Later, the *conceptReference* can be used to semantically map an event in pattern properties to a Web service operation and vice versa.

Expression, Unary and Binary: *Expression* is a general class for patterns and operations. For the succinctness of the PROPOLS ontology, all the patterns and scopes are the subclasses of *Unary* or *Binary*. So they share the properties defined in Unary or Binary, either has an *operand* property pointing to *Expression* or has a *firstOperand* property and a *secondOperand* property. For example, *Exists* is a unary pattern and *Between* is a binary scope. Further restrictions are set for different kinds of expressions. For example, a scope only accepts operations as its operands.
ConstriantList:. This class is a container for pattern properties/constraints.

4.2   Composite Patterns

As shown in Fig. 3, a composite pattern is the composition of pattern properties using Boolean logic operators including *Not*, *And*, *Or*, *Xor* and *Imply*.

Fig. 3. Composite Pattern Structure

With composite patterns, complex properties can be stated straightforward. For example, if we want to define a property according to "If the order is fulfilled, the bank ensures that the payment transfers to AdvantWise". We may write a composite pattern connected with the 'And' operator:

`Customer.GetOrderFulfilled` **`Precedes`** `Bank.Transfer` **`Globally`** **`And`** `Customer.GetOrderFulfilled` **`LeadsTo`** `Bank.Transfer` **`Globally`**	(1)

In this situation, both 'GetOrderFulfilled' and 'Transfer' should occur and must occur sequentially in the program. This composite pattern is used frequently, so we add a stereotyped composite pattern: *PLeadsTo,* to our pattern system.

Another example of complex properties is the first requirement of the example scenario. This requirement claims an 'exclusive or' relation between order confirmation and rejection. It also demands a precedent occurrence of the "check order" activity. With composite patterns, we may specify this requirement as follows:

`(Customer.ConfirmOrder` **`Exists Globally`** **`Xor`** `Customer.RejectOrder` **`Exists Globally`**`)` **`And`** `Manufacturer.CheckOrder` **`Precedes`** `Customer.ConfirmOrder` ` ` **`Globally`** **`And`** `Manufacturer.checkOrder` **`Precedes`** `Customer.RejectOrder` **`Globally`**	(2)

Next, we briefly explain the semantics of composite patterns. A formal semantics definition can be found in an accompanying technical report [15].

As stated earlier, every elementary pattern property has a corresponding FSA semantics. We thus define the semantics of a composite pattern property from the logical composition of FSAs of its component pattern properties. For example, Fig. 4 shows the resultant FSAs of 4 logical compositions between two properties: "P1 **exists globally**" and "P2 **exists globally**". The states are the Cartesian product of the two property FSAs' states. The first number in a state label represents the state of the first property FSA, while the second represents the state of the second property FSA. The final states of the composite pattern are determined by the logic operator used. For example, the pairing of one final state 'And' one non-final state is a non-final state, and one final state 'Xor' one non-final state is a final state. The final states for different compositions are also described in Fig. 4.

Logic Operator	Final States
And	11
Or	01,10,11
Xor	01,10
Imply	00,01,11
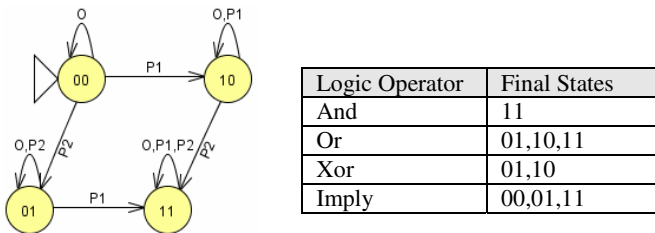
Fig. 4. Logical Compositions of Two 'Exists' Properties

With such definitions, we have proved in [15] a theorem: the language set that can be accepted by a composite FSA is the composition of the language set of the component elementary FSAs. For example, if we have a composite pattern 'pr1 And pr2', then Language(pr1 **And** pr2) = Language(pr1) **And** Language(pr2). Here, the '**And**' between two sets Language(pr1) and Language(pr2) is the set-intersection operator. This theorem proves the correctness of our semantics definition to composite patterns.

4.3   PROPOLS Example

In this subsection, we use PROPOLS to describe the example requirements introduced in section 2. For easy reading, we first write the pattern properties in a simple text format in Fig. 5.

In Fig. 5, the first property is for requirement 1. The second, third, and fourth constraints are for requirement 2. The last property is for requirement 3. These pattern properties are quite intuitive and self-explanatory.

```
Constraint List: Hard Credit Rule
1.   (See property (2) in section 4.2)
2.   Customer.ConfirmOrder PLeadsto
      Bank.Deposit Globally
3.   Bank.Depoist PLeadsto
      Manufacturer.StartOrderProcessing Globally
4.   Customer.GetOrderFulfilled PLeadsto
      Bank.Transfer Globally
5.   Manufacturer.CheckInventory Precedes
      Manufacturer.ScheduleProducing Globally
```

Fig. 5. Pattern Properties for the Example Requirements

To showcase the real definition of properties, we present part of the PROPOLS code for the first property in Fig. 6. A complete version can be found in [15].

```
<And rdf:ID="And_Property">
  <secondOprand rdf:resource="#Pre_Canc"/>
  <firstOprand rdf:resource="#And_inst_1"/></And>
<Precedes rdf:ID="Pre_Canc">
  <scope><Global rdf:ID="Global_Inst"/></scope>
    <firstOprand>
      <Operation rdf:ID="checkOrder">
       <provider rdf:datatype="&XMLSchema;anyURI">
        #Manufacturer</provider>
      <conceptReference rdf:datatype="&XMLSchema;anyURI">
        #CheckOrder</conceptReference>
      </Operation></firstOprand>
   <secondOprand rdf:resource="#RejectOrder"/>
</Precedes>
……
```

Fig. 6. Excerpt of an Example PROPOLS Property Definition

5   Verification of BPEL

In this section, we present an approach to the compliance checking of BPEL schemas against PROPOLS properties. The presentation starts with a high-level description of a BPEL schema to implement AdvantWise's online purchase process in the aforementioned example scenario, and then explains the verification approach using the example BPEL schema. A snapshot of our developed verification tool is also presented.

5.1   An Example BPEL Schema

Fig. 7 shows the main structure of the BPEL schema as an implementation of the online purchase process for AdvantWise. This diagram keeps the message exchange primitives in the actual BPEL schema. It uses black bars to represent the other participants of the process, including the Customer and the Bank.
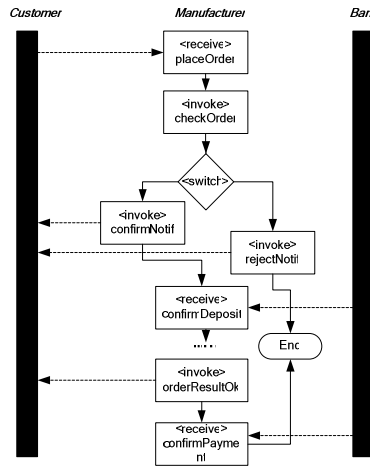
Fig. 7. Example BPEL Schema: Main Structure

5.2   Verification Approach

Fig. 8 illustrates our approach to verifying the compliance of BPEL schemas to PROPOLS properties. The main steps of the verification process are explained below.
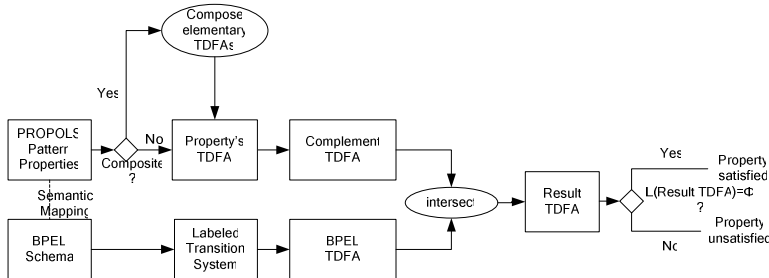

**Fig. 8.** Principle of PROPOLS Verification Approach

Semantic Mapping. First of all, there is a semantic mapping between the operations defined in PROPOLS and Web service operations. The mapping is based on the condition that both of these operations refer to the same concept in the domain ontology. The PROPOLS operations can tell their semantics via the *conceptReference* property. There are two typical ways in which one can add semantic annotations to Web service operations: by using an external annotation file or directly appending semantic annotations to the WSDL file. In this context, we use the latter approach. As exemplified in Fig. 9, we extend the *Customer* Web service's operation definition with WSDL-S semantic extension element *wssem:modelReference*.

```
<portType name="Customer>
  <operation name="confirmNotif"
    wssem:modelReference="
    http://www.purl.org/onto/operationOnto#ConfirmOrder>
    …
   <operation name="rejectNotif"
     wssem:modelReference="
     http://www.purl.org/onto/operationOnto#RejectOrder>
 …
```

Fig. 9. Example Semantic Annotations to Web Service Operations in WSDL

In the above example, the Web service operation *confirmNotif* and PROPOLS operation *ConfirmOrder* refer to the same ontology concept, so we can use *confirmNotif* to replace *ConfirmOrder* in the compliance checking process. A complete treatment of the mapping between operations can be found in [15].

Verification Process. As shown in Fig. 8, the verification is conducted in 3 steps. (1) For every pattern property, a semantic equivalent Total and Deterministic FSA (TDFA) is built. If the property is a composite one, the corresponding TDFA is constructed by composing the TDFAs of its sub-properties according to the composition semantics definition in section 4.2. (2) For the BPEL schema, a finite and deterministic LTS model is generated, from which a TDFA is built by introducing the set of final states and an error state to collect all the unacceptable events (or operation invocations) of each state. (3) The compliance of the BPEL schema to the PROPOLS properties is then checked as a verification problem of whether the accepting event sequences of the BPEL TDFA are all present in the accepting event
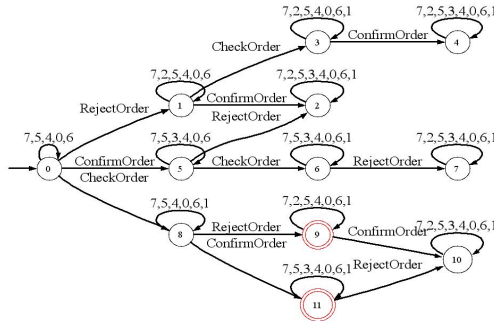
Fig. 10. TDFA of the Example Property

sequence set of the property TDFA. This is done by testing the emptiness of the intersection of the BPEL TDFA and the complement of the property TDFA. A detailed explanation of this approach can be found in [15].

Next, we illustrate the verification process using property 1 in Fig. 9. Fig. 10 shows property 1's TDFA. Fig. 11 shows the TFDA of our example BPEL schema. This TFDA was obtained in two steps. First, Foster's BPEL2LTS tool [9] is used to obtain a LTS for the BPEL schema. Then, the LTS is transformed to a FSA (See [15] for detail).
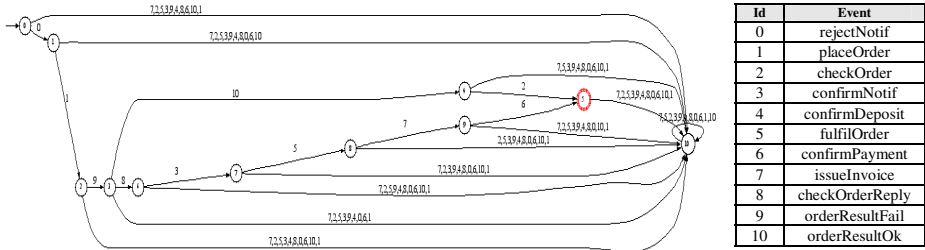

Id	Event
0	rejectNotif
1	placeOrder
2	checkOrder
3	confirmNotif
4	confirmDeposit
5	fulfilOrder
6	confirmPayment
7	issueInvoice
8	checkOrderReply
9	orderResultFail
10	orderResultOk

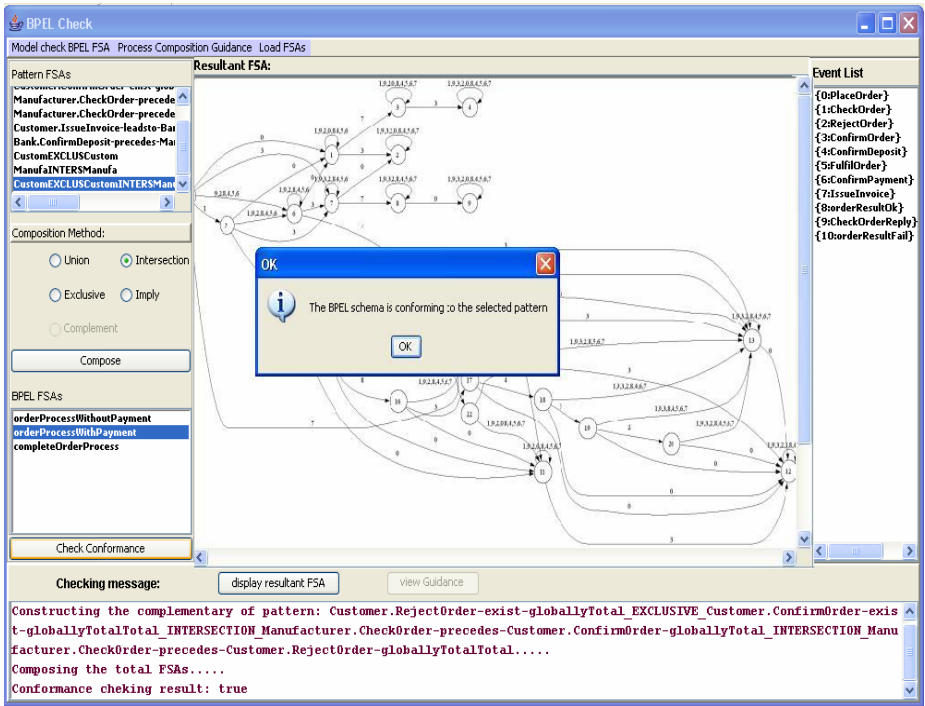**Fig. 11.** TDFA of the Example BPEL Schema

Fig. 12. Snapshot of the Verification Example

After obtaining the complement of the property TDFA, it is intersected with the BPEL TDFA. It turned out that the resultant FSA has no final state. This means that the BPEL schema conforms to the example pattern property. A snapshot of the final result is shown in Fig. 12.

We have implemented a prototype tool for verifying BPEL Schemas against PROPOLS properties. Fig. 12 is the snapshot of this tool. The left panel contains a list of properties, the logical operators, and the BPEL schema waiting for verification. While composing a property and a BPEL schema according to the above-stated approach, the resultant TDFA is shown in the right panel.

6   Related Work

Particularly relevant to the work presented in this paper are two streams of work: formal verification of BPEL schemas and property specification patterns.

A body of work has been reported in the area of formal property specification and verification of BPEL schemas. The main differences among them lie in the property specification language and the formal semantic model for BPEL. In [16], Foster relies on Finite State Processes (FSPs) to both semantically represent a BPEL schema and specify the properties. In [17], Stahl maps BPEL schemas into Petri nets and utilises a verification tool LORA (Low Level Petri net Analyzer) to verify CTL properties. In [18], the authors map BPEL into Promela, the input language of the model checker SPIN, and then use SPIN to check LTL properties. The most significant difference between these approaches and our work is that we focus on a practitioner-oriented approach to property specification. Compared to their heavy-weighted formal property specification languages, our pattern-based PROPOLS language is easier to understand and use. Also, its ontology-based nature helps in establishing a natural connection between pattern-based properties and the domain knowledge.

The property specification patterns are originally proposed by Dwyer et al. in [6]. Since then they have been applied and extended in many ways. Smith et al. [7] proposed a specification approach which enables fine-tuning patterns to achieve more precise meanings, based on a combined use of a "disciplined" natural language and a FSA template language. For example, six different templates were identified to fine-tune the response pattern. Gruhn et al. [8] extended the patterns with time for describing real-time related properties. Paun et al. [19] extended the patterns to deal with events in a state-based formalism. Furthermore, the patterns are the foundation for the extensible specification language in the Bandera system [20].

7   Conclusion

In this paper, we proposed a verification approach to BPEL schemas, which employs an ontology language PROPOLS for the property specification. PROPOLS builds on and extends Dwyer et al.'s pattern system. Its pattern-based nature enables software practitioners to write formal behavioral properties more easily. Its logical composite pattern mechanism allows one to state complex requirements. The ontology encoding also facilitates the sharing and reuse of PROPOLS constraints.

In the future, we intend to develop a graphical interface for the PROPOLS language. We also intend to apply the pattern properties in PROPOLS to provide just-in-time guidance to the BPEL designer during the service composition process.

References

1. Papazoglou, M.P., Georgakopoulos, D.: Special Issue on Service Oriented Computing. Communications of ACM 46 (10) (2003) 24 – 28
2. Alonso, G., Casati, F., Grigori, Kuno H., Machiraju, V.: Web Services Concepts, Architectures and Applications. Springer-Verlag (2004)
3. Arkin, A., Askary, S., Bloch, B., Curbera, F., Goland, Y., Kartha, N., Liu, C.K., Thatte, S., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0 Working Draft. http://www.oasis-open.org/committees/download.php/10347/wsbpel-specification-draft-120204.htm (2004)
4. BPMI: Business Process Modeling Language. http://www.bpmi.org/ (2002)
5. Clarke, E.M., Moon, I., Powers, G.J., Burch, J.R.: Automatic Verification of Sequential Control Systems using Temporal Logic. American Institute of Chemical Engineers Journal, 38 (1) (1992) 67 – 75
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property Specification Patterns for Finite-State Verification. In 2nd Workshop on Formal Methods in Software Practice (1998) 7 - 15, Clearwater Beach, FL, USA
7. Smith, R.L., Avrunin, G.S., Clarke L.A., Osterweil L.J.: PROPEL: An Approach Supporting Property Elucidation. In Proc. 24th International Conference on Software Engineering (2002) 11-21, Orlando, FL, USA
8. Gruhn V. Laue R.: Specification Patterns for Time-Related Properties. In 12th International Symposium on Temporal Representation and Reasoning (2005) 189 - 191, Burlington, Vermont, USA
9. Foster, H.: LTSA WS-Engineering. http://www.doc.ic.ac.uk/ltsa/bpel4ws/ (2006)
10. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in Property Specifications for Finite state Verification. In Proc. International Conference on Software Engineering (1999) 411-420, Los Angeles, CA, USA
11. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: A System of Specification Patterns. http://www.cis.ksu.edu/santos/spec-patterns, (1997)
12. Jin, Y., Han, J.: Consistency and Interoperability Checking for Component Interaction Rules. In Proc. 12th Asia-Pacific Software Engineering Conference (2005), Taipei, Taiwan
13. Li, Z., Han, J., Jin, Y.: Pattern-Based Specification and Validation of Web Services Interaction Properties. In Proc. 3rd International Conference on Service Oriented Computing (2005) Amsterdam, Netherland
14. OntoViz Tab: Visualizing Protégé Ontologies http://protege.stanford.edu/plugins/ontoviz/ontoviz.html (2005)
15. Yu, J., Phan, M.T., Han, J., Jin, Y.: Pattern based Property Specification and Verification for Service Composition. Technical Report SUT.CeCSES-TR010. CeCSES, Swinburne University of Technology, http://www.it.swin.edu.au/centres/cecses/trs.htm (2006)

16. Foster, H.: A Rigorous Approach to Engineering Web Services Compositions. PhD thesis, Imperial College London. http://www.doc.ict.ac.uk/~hf1 (2006)
17. Stahl C.: A Petri Net Semantics for BPEL. Informatik-Berichte 188, Humboldt-Universitat zu Berlin, June 2005 (2005)
18. Fu, X., Bultan T., Su J.: Analysis of Interacting BPEL Web Services. In Proc. 13th World Wide Web Conference (2004) 621-630, New York, NY, USA
19. Paun, D.O., Chechik, M: Events in Linear-Time Properties. In Proc. 4th International Conference on Requirements Engineering (1999) Limerick, Ireland
20. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Laubach', S., Pasareanu, C.S., Zheng, R., Zheng, H: Bandera: Extracting finite-state models from Java source code. In Proc. 22nd International Conference on Software Engineering (2000) 439-448, Limerick, Irelan