Streamlining Pattern Support Assessment for Service Composition Languages

Jörg Lenhard, Andreas Schönberger, and Guido Wirtz

Distributed and Mobile Systems Group, University of Bamberg, Germany
{joerg.lenhard,andreas.schoenberger,guido.wirtz}@uni-bamberg.de

Abstract. Various process modeling formalisms have been leveraged to specify service compositions. For assessing the expressiveness of similar languages and for providing best practice knowledge, patterns have frequently been proposed. However, the pattern catalogs proposed do not all share and document the criteria that were used for assessing pattern support. Furthermore, the scaling of the support measure frequently is very coarse, only providing a basic level of selectivity. This paper proposes an approach that allows for measuring the pattern support for different catalogs in a uniform manner. The selectivity of the support measure is improved by using the edit distance for calculating its degree. The feasibility of the approach is shown by preliminary results of the analysis of selected patterns and orchestration languages.

Keywords: SOA, Pattern, Service Composition Language, Edit Distance, Orchestration

1 Introduction

A powerful property of *service-oriented architectures* (SOAs) is the service composition layer [9]. This layer covers the construction of composite services from other services which is often achieved by combining calls to existing services in a process-based manner. This essentially involves the definition of control- and data-flow dependencies between the different service invocations. Representations of process-based service composition languages are choreography and orchestration languages [10]. Traditional notions such as Turing-completeness are inappropriate for capturing the suitability of service composition languages. In the area of workflow systems, describing reasonable aspects of languages in the form of *patterns* and analyzing existing languages for their support for those patterns was proposed. This approach was initiated by the *workflow patterns initiative* [16] and was widely used by product vendors and scientific research from its start on. Today, many different pattern catalogs are available. However, a study that analyzes a language using multiple pattern catalogs faces several problems. Differences among the various publications according to what constitutes which level of support limit comparability. In fact, most authors use different notions of what counts as support and also do not document clearly what criteria need to be fulfilled by a candidate solution to offer support for a pattern. This way, the degree of support determined sometimes seems to be based on personal bias.

The intent of the support measure is to describe how directly or easily a pattern can be implemented in a language using built-in constructs. It does generally *not* state whether or not a pattern can be implemented in a language at all. The degree of support states to what extent the user of a language is aided by the constructs directly available by or built into the language. Its scaling typically is trivalent (or in some cases such as [8, 18] quadrivalent) and distinguishes whether a solution provides *direct* (+), *partial* (+/-) or *no direct support* (-) for a pattern ([16], p. 50), based on the amount of constructs needed in a solution. Constructs are the core building-blocks of a language, such as a decision activity or a fork activity. Adjacent concepts, such as variables or correlation sets do generally not count as constructs. Usually, a candidate solution that uses only a single construct provides direct support. A combination of two constructs results in partial support and if more than two constructs are needed no direct support is provided. This trivalent degree can be too coarse. For example, consider the case where a pattern is directly supported in two languages by a single construct. In language A, the single construct can be used in a straight-forward manner and the solution to the pattern is complete. In language B, the single construct needs to be used and a complex configuration of the construct is necessary, consisting of, say, three changes to the default values of its attributes which may be interdependent on each other. Furthermore, the creation of a variable in the process model is also needed. Obviously, the solution in language A is more direct than the solution in language B. Nevertheless, they are equal concerning their degree of support.

This paper tackles these problems of comparability and selectivity by proposing a unified approach for determining the degree of support a given solution provides for a pattern. This approach is derived from the different methodologies used by the authors of relevant catalogs. It works in two steps:

1. For a given candidate solution, it is first determined whether it provides a *valid implementation* for a given pattern.
2. If so, the degree of support it provides is calculated. This calculation is done using an alternative scaling of the support measure, the *edit distance* based on high level change operations, to enhance the selectivity of the results.

The following section briefly describes relevant pattern catalogs and related analyses. Section 3 outlines the proposed approach, followed by preliminary results for the support of two orchestration languages for selected patterns in Sect. 4. Section 5 concludes.

2 Related Work

The workflow patterns initiative published several pattern catalogs, most notably the *control-flow patterns* [13, 16]. Other aspects of workflows are covered by the *data patterns* [12] and the *resource patterns* [11]. [14] also presents mechanisms for *exception handling* in the form of a pattern catalog. The *service interaction patterns* [2] were the first pattern catalog which is specific for languages focusing on service-based processes and describe common interaction scenarios. [1] followed this catalog with a set of patterns that capture *correlation* mechanisms. [17, 18] consider patterns for dealing with *changes* to processes in *process-aware information systems*. Like the service interaction patterns, [15] describes functions that are common to business processes in the form of *activity patterns*. [3] addresses ways in which process instances can be created in the form of *process instantiation patterns*. Recently, also *time patterns* [7, 8] for analyzing the support for time constraints in a language have been proposed.

With the exception of the activity patterns, all these publications of pattern catalogs do also provide an analysis of the support of selected languages for the patterns of the catalog. They value the validity of possible solutions using criteria specific for the catalog. [1,3,7,8,13,16–18] do present a tri- or quadrivalent scaling of the support measure. [2, 14] simply state whether a pattern can be realized at all. The edit distance presented here relates to the *graph-edit distance* [5]. This edit distance is used in [17] based on editor operations for demonstrating the necessity for the support for adaptation patterns. Here, we use a more specific set of edit operations based on the structure of a language to measure the degree of pattern support.

Based on the pattern catalogs, there are also a variety of studies that perform additional analyses. In the area of orchestration languages, the *Web Services Business Process Execution Language* (BPEL) is analyzed in the context of the control-flow, data, resource, service interaction, correlation and process instantiation patterns (cf. above). [4, 19] also compare the expressiveness of BPEL to other Web Services composition languages. Both of the studies use several pattern catalogs. An alternative orchestration language, *Windows Workflow* (WF), is examined for its support of control-flow patterns in [21] to provide initial insights into its control-flow expressiveness and is compared to BPEL. This paper proposes a two-step approach to improve such analyses comprising multiple pattern catalogs which increases comparability and provides a higher level of selectivity.

3 Approach

The first step of the approach is to determine whether a given candidate solution forms a valid implementation of a given pattern. Only a solution that fulfills this minimal criterion is able to provide support for a pattern. The decision whether this is the case is based on the structure and components of a pattern which are similar for all pattern catalogs at hand, although not all of the catalogs contain all of the aspects discussed below. Five components of a pattern are essential for determining the validity of an implementation.

Pattern description: The pattern description specifies the nature of the pattern and its *core aspects*. To provide a valid implementation, a candidate solution must cover all core aspects that are found in the pattern description, as explicitly stated in [2]. This minimum component can be found in any pattern catalog.

Pattern context: The *context*, in some cases called *issues* [15], describes several assumptions or criteria about the environment in which a pattern operates. To provide a valid implementation, at most one of these criteria may not be met by a candidate solution. This pays tribute to the fact that the support for a pattern should still be calculated even if minor aspects cannot be covered. These constraints can be inferred from the evaluation criteria of [11–13]. As an example, the *Structured Synchronizing Merge* pattern requires the existence of preceding *Multi-Choice* construct in a context criterion ([13], pp. 17 - 19). Context criteria can be found in [2, 8, 11–13, 15, 16].

Execution traces: Closely related to the pattern context is the notion of *execution traces* ([8], p. 98). An execution trace defines the structure of all possible execution sequences of activities that are valid for a given pattern. Examples are mathematical expressions, used in [1, 8, 17], or graphical notations such as Petri Nets, used in [13, 15]. If a formalization for execution traces is present, a candidate solution must also conform to these traces which is explicitly stated in [8, 17].

Design choices: In most cases, the definition of a pattern is flexible to some extent. Certain aspects are left open to the choice of the implementer of a pattern, which are described as *design choices* ([8], p. 97). Each design choice denotes a list of alternative aspects one of which can be chosen when implementing a pattern. A combination of different aspects from the design choices attached to a pattern then forms a *pattern variant* ([8], p. 97). A candidate solution must implement at least one pattern variant (cf. [8, 18]), omitting at most one of the design choices of the variant. As an example, a solution for the *Durations* pattern that supports only maximum, but not minimum durations of activities still forms a valid implementation of the pattern ([8], p. 100). Design choices can be found in [2, 8, 15, 18].

Data types: A pattern might inherently depend on the availability of specific data types, such as dates or timestamps [1, 7, 8]. To provide support for a pattern, a corresponding data type must be available in a language. Additionally, if needed in a candidate solution, necessary operations for comparing or manipulating these types must be provided, as can be found in the evaluation contained in [7].

For a candidate solution that provides a valid implementation, the degree of support can be calculated. As shown in Sect. 1, the traditional trivalent degree can be too coarse. This situation can be improved by relying on an alternative measure. The problem of qualifying the effort needed to realize a pattern is basically a question of *distance* between processes. Say process X is a process stub without specific functionality and process Y is an extension of X that adds exactly the solution of a pattern. The less distant X is to Y, the less effort is needed to transform X into Y. So, the support for a pattern in a language can also be measured by computing the distance between two processes written in the language, where one of the processes extends the other one with the implementation of a given pattern. Listing 1 outlines such a process stub for BPEL. It contains necessary import definitions and the definition of one `partnerLink` which is inevitable for a working process. The control-flow of the minimal process is formed by a `receive` activity that creates a new process instance and uses a `variable` as input embedded in a `sequence` activity. The pattern implementation then succeeds the `receive` activity.

Listing 1. Process stub for BPEL

```
<process>
   <import location="ProcessInterface.wsdl" />
   <partnerLinks>
       <partnerLink name="MyPartnerLink" myRole="patternRole" />
   </partnerLinks>
   <variables>
       <variable name="StartProcessInput"/>
   </variables>
   <sequence>
        <receive createInstance="yes" variable="StartProcessInput"
         partnerLink="MyPartnerLink" operation="StartProcess"/>
         <!--Pattern Implementation-->
   </sequence>
</process>
```

[20] presents several measures for computing the similarity between process models. A foundation for these similarity measures that seems very applicable for the problem at hand is the *edit distance*. This distance measures the smallest distance between two strings by calculating the minimum number of change operations, being substitutions, insertions or deletions of characters that are needed to transform one string into another. For the problem at hand, the basis are of course process models and not strings. The models to be compared are a process stub, as demonstrated in List. 1 and a process extending this stub with the implementation of a pattern. Counting substitutions of characters would make no sense here, as the distance in concepts and constructs would get lost in syntactical noise. For example a language could tend to have higher distances simply because its activities have longer names. Much more applicable in this case are high level changes to the structure of the process model, as opposed to changes of characters. The difference is that high level changes comprise larger structures and satisfy minimalistic semantical constraints. Examples are the insertion of an activity and the setting of its name, the insertion of a variable and the setting of its name and type or the setting of a target variable and expression in an assignment. A concrete example for BPEL would be the configuration of correlation for a `receive` activity. This involves the creation of a `correlations` and a `correlation` element, the setting of its name and potentially whether the correlation set should be initiated. Counting each syntactical modification, instead of the single high level operation *add correlation to receive*, adds noise to the final result. The intent of the edit distance here is after all not to capture differences in naming, but differences in concepts and constructs, because these differences better describe the effort needed by the user of a language. The edit distance can now be calculated by adding up the amount of such high level insertions, substitutions and deletions needed. Using the same set of high level changes

as basis for the edit distance in the assessment of different catalogs ensures comparability between the results. Generalizing the set of high level changes and making it applicable for different languages also provides comparability between the languages, even for different pattern catalogs.

Obviously, such edit operations can be facilitated by using a sophisticated integrated development environment. The aim of this study however, is to measure the support provided by a language and not by tools available for the language. The edit distance as discussed here abstracts from the availability of specific tools that facilitate edit operations. The same applies to the representation of the language [6]. The identification of constructs that add to the edit distance cannot easily be automated by relying on the syntactical elements of a representation format such as XML tags or state machine nodes.

4 Preliminary Results

Table 1 shows the results for an analysis of the support of the two orchestration languages WS-BPEL 2.0 and WF 4 for selected pattern groups of the control-flow [13, 16], the service interaction [2] and the time patterns [7, 8]. WF 4 represents

Table 1. Support of selected patterns. Edit distance is listed first, followed by trivalent scaling in parentheses.

Pattern	WF 4	BPEL 2.0
State-based Control-flow Patterns [13, 16]		
Deferred Choice	9 (+/-)	8 (+)
Interleaved Parallel Routing	- (-)	12 (+/-)
Milestone	11 (+/-)	11 (+/-)
Critical Section	9 (+/-)	11 (+/-)
Interleaved Routing	9 (+/-)	11 (+/-)
Multi-transmission Service Interaction Patterns [2]		
Multi Responses	71 (-)	90 (-)
Contingent Requests	28 (+)	38 (+)
Atomic Multicast Notification	40 (-)	- (-)
Recurrent Process Elements Time Patterns [7, 8]		
Cyclic Elements	12 (+/-)	- (-)
Periodicity	8 (+/-)	7 (-)

the Windows Workflow Foundation in revision 4 (`http://msdn.microsoft.com/en-us/netframework/aa663328.aspx`). The analysis is performed using the approach of the previous section. The process stub used for WF 4 is semantically identical to the one used for BPEL (cf. List. 1) and consists of a `Receive` activity that creates a new process instance embedded in a `Sequence` activity. There is currently no study that measures the degree of the pattern support of WS-BPEL 2.0 and WF 4 for these pattern catalogs. For comparison, we computed the degree of support using the edit distance and the trivalent measure. The edit distance is shown first followed by the trivalent measure in parentheses. A value of '-' for the edit distance means that no valid solution could be found in the scope of the language. As opposed to this, a value of '-' for the trivalent measure means that either no valid solution could be found or that all possible valid solutions require the use of more than two constructs. The table shows that the edit distance allows for a better distinction. In several cases, both languages have the same degree of support with the traditional measure, while the edit distance unveils the differences.

As an example, the realization of the Deferred Choice pattern ([13], pp. 33/34) is outlined in List. 2. The pattern is realized using two `onMessage` activities embedded in a `pick` activity. The following steps are necessary to realize a valid implementation: (i) replace `receive` with `pick`; (ii) set `createInstance` attribute of `pick` to `yes`; (iii) create first `onMessage`; (iv) configure messaging properties of first `onMessage`, consisting of the setting of the `partnerLink`, `portType` and `operation`; (v) embed `empty` in first `onMessage` (an `onMessage` must contain a child activity); (vi - viii) create and configure the second `onMessage` similar to the first one. In summary, the edit distance of the solution amounts to eight.

Listing 2. Realization of Deferred Choice pattern in BPEL

```
<pick createInstance="yes">
    <onMessage partnerLink="MyPartnerLink" operation="Choice1">
        <empty />
    </onMessage>
    <onMessage partnerLink="MyPartnerLink" operation="Choice1" >
        <empty />
    </onMessage>
</pick>
```

5 Conclusion and Future Work

This work presents an improvement to the method of pattern-based analysis aiming at a higher degree of comparability between different pattern catalogs and a higher level of selectivity of the results. The comparability between pattern catalogs can be improved by using a unified approach for determining whether a given candidate solution provides a valid implementation of a pattern. The approach presented here essentially unites the methodologies used by other authors and insights gained during own analyses. The level of selectivity can be increased by using an alternative support measure, the edit distance based on high level changes. Preliminary results show the applicability of the approach. The next step is to test the approach in a complete study comparing several languages and multiple pattern catalogs which is ongoing work. Especially, languages with a focus on expressiveness such as YAWL might bear interesting results. Also, a better formalization of several pattern catalogs in terms of execution traces would be beneficial.

References

1. A. P. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *FASE*, Braga, Portugal, March/April 2007.

2. A. P. Barros, M. Dumas, and A. H. M. ter Hofstede. Service Interaction Patterns. In *BPM*, pages 302–318, Nancy, France, September 2005.
3. G. Decker and J. Mendling. Process Instantiation. *DKE, Elsevier*, 68:777 – 792, 2009.
4. G. Decker, H. Overdick, and J. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *EMISA*, Hamburg, Germany, October 2006.
5. R. M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph Matching Algorithms for Business Process Model Similarity Search. In *BPM*, Ulm, Germany, September 2009.
6. O. Kopp, D. Martin, D. Wutke, and F. Leymann. The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages. *EMISAIJ, GI e.V.*, 4:3 – 13, 2009.
7. A. Lanz, B. Weber, and M. Reichert. Time Patterns in Process-aware Information Systems - A Pattern-based Analysis - Revised version. Technical report, University of Ulm, Germany, 2009.
8. A. Lanz, B. Weber, and M. Reichert. Workflow Time Patterns for Process-Aware Information Systems. In *BPMDS and EMMSAD in conjunction with CAiSE*, 2010.
9. M. P. Papazoglou and D. Georgakopoulos. Service-oriented Computing. *Communications of the ACM*, 46(10):24–28, October 2003.
10. C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, October 2003.
11. N. Russell, A. H. M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. In *CAiSE*, pages 216–232, Porto, Portugal, June 2005. Springer.
12. N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In *ER*, Klagenfurt, Austria, October 2005. Springer.
13. N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. Technical report, BPM Center Report, 2006.
14. N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede. Workflow Exception Patterns. In *CAiSE*, pages 288–302, Luxembourg, Luxembourg, June 2006.
15. L. H. Thom, M. Reichert, and C. Iochpe. Activity Patterns in Process-aware Information Systems: Basic Concepts and Empirical Evidence. *IJBPIM*, 4(2):93–110, 2009.
16. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases, Springer*, 14(1):5–51, 2003.
17. B. Weber, S. Rinderle, and M. Reichert. Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical report, University of Twente, 2007.
18. B. Weber, S. Rinderle, and M. Reichert. Change Patterns and Change Support Features - Enhancing Flexibility in Process-Aware Information Systems. *DKE, Elsevier*, 66:438–466, July 2008.
19. P. Wohed, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In *ER*, Chicago, Illinois, USA, October 2003.
20. A. Wombacher and C. Li. Alternative approaches for workflow similarity. In *IEEE SCC*, Miami, Florida, USA, July 2010.
21. M. Zapletal, W. M. P. van der Aalst, N. Russell, P. Liegl, and W. H. An Analysis of Windows Workflow's Control-Flow Expressiveness. In *ECOWS*, pages 200 – 209, Eindhoven, The Netherlands, November 2009.