A Model-Driven Approach for QoS Prediction of BPEL Processes

Jiangxia Wu and Fangchun Yang

State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications, 100876 Beijing, China
wujiangxia@gmail.com, fcyang@bupt.edu.cn

Abstract. Business Process Execution Language (BPEL) is one of the most popular languages for Web service composition. To predict the QoS of composite service processes specified in BPEL gives the way to tell whether the process meet the non-function requirements, and to choose the process with better QoS from those with similar function. A model-driven approach for QoS prediction of BPEL processes is proposed in this paper, which has a two-layer architecture: One is the prediction model specifying necessary information for prediction and independent of specific languages, and the other is the semantic model of specific languages such as BPEL. A set of transformation rules is defined between the two layers so that processes specified in specific languages can be transformed to the prediction model. A prediction algorithm is defined based on the prediction model, and through the algorithm the average value of process QoS attribute can be computed. The approach can be used not only to BPEL processes but also to processes in other specifications such as BPML and BPSS, if the mapping rules between the semantic models of these languages and the prediction model are defined. The feasibility and good accuracy of the approach has been proved by the experiment.

Keywords: QoS prediction, Web service composition, BPEL, model-driven.

1 Introduction and Related Work

Business Process Execution Language (BPEL) is one of the most popular languages for Web service composition. To predict the QoS of BPEL processes gives the way to tell whether the process meet the non-function requirements, and to choose the process with better QoS from those with similar function. The research in QoS prediction of Web service composition is in its infancy. Zeng introduced the aggregation function approach for prediction [1]. The function is rather simple and has limited accuracy. Grassi proposed the Software Architecture based approach [2] which is based on the research of component system prediction and is concerned with the execution environment of Web services. Therefore, it is not feasible for Web service composition. Chadrasekaran introduced a simulation based approach [3] which relies on specific simulation tool. And the Workflow Based Approach (*WBA*) [4, 5] is proposed by Cardoso and Jaeger respectively and has better feasibility and accuracy than others methods. However, neither of the existing methods can be used

D. Georgakopoulos et al. (Eds.): ICSOC 2006 Ws, LNCS 4652, pp. 131–140, 2007.

for the prediction of BPEL processes, because they are not based on the semantic concepts of BPEL.

In this paper, we propose a model-driven approach for BPEL process prediction, which introduces a two-layer architecture: One is the prediction model specifying necessary information on composite services for prediction and independent of specific languages, and the other is the semantic model of specific languages such as BPEL. A set of transformation rules is defined between the two layers so that processes specified in specific languages can be transformed to the prediction model. And a prediction algorithm is defined based on the prediction model, by which the average value of QoS attributes of processes can be computed. The approach can be used for not only BPEL processes QoS prediction but also processes specified in any other languages such as BPML, WSCI etc., if the mapping rules between the semantic models of these languages and the prediction model are defined. The feasibility and good accuracy of the approach has been proved by the experiment.

The rest of the paper is organized as follows: Section 2 gives the formal definition of the prediction model. Section 3 proposes the transformation rules between semantic model of BPEL and the prediction model. Section 4 introduces the prediction algorithm and the analysis of the time complexity. The experiments and analysis are given in section 5. Finally, we conclude in section 6.

2 Prediction Model

2.1 Model-Driven Prediction

Model-Driven Architecture (*MDA*) is promoted by the Object Management Group (*OMG*) for software development [7]. The main idea of MDA is to achieve portability, interoperability, and reusability through an architectural separation and transformation of concerns between the design and implementation of software. The work described in this paper adopts the MDA strategy to predict the QoS of composite service processes specified in BPEL.

Model-driven QoS prediction is based on a two-layer architecture shown in figure. 1. One is the prediction model specifying necessary information on composite services for prediction and independent of specific languages, and the other is the semantic model of specific languages such as BPEL. A set of transformation rules is defined between the two layers so that processes specified in specific languages can be transformed to the prediction model. Based on the prediction model, a prediction

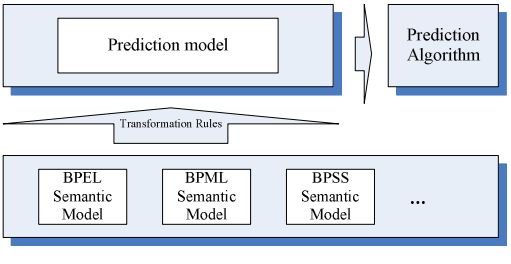

Fig. 1. The architecture of model-driven prediction

algorithm is defined to compute the average value of QoS attributes of processes specified in the prediction model. Thus, the approach can be used to predict the process in any languages such as BPML, BPSS etc., if the mapping rules between the semantic model of the language and the prediction model is defined.

2.2 Model Definition

The prediction model is the abstraction of information elements necessary for QoS prediction of a composite services process. It is independent of specific languages so it can be used for the prediction of process in any specification including BPEL. The model is made up of the following concept definitions.

Definition 1. Composite service process Γ. Γ is a 4-tuple with the format of (*CP*, *T*, *C*, *Q*). It is used to represent the composite service process to be predicted. Γ can be specified as a workflow [8]. And the tasks in workflows represent the service invocation operations, and the transitions between tasks represent the orders between invocation operations. The definition of the four elements of Γ is given below.

Definition 2. Composition pattern set (*CP*). *CP* represents the set of composition patterns which constitute the composite service process. Composition pattern is the abstraction of basic architecture of composite services [5, 6], which is composed of a set of tasks and defines the execution order of the tasks and the completion symbol of the pattern. There are seven types of composition patterns *Sequence*, *Loop*, *XOR_XOR*, *AND_AND*, *AND_DISC*, *OR_OR* and *OR_DISC*. The definition of each type is given in table 1. Composition patterns can be nested and sub-patterns can be treated as the tasks of the parent-pattern.

Table 1. Definition of composition pattern types

cp Type	Definition
Sequence	Containing n (n>1) tasks which are executed in sequence.
Loop	Containing 1 task which is executed repeatedly.
XOR_XOR	Containing n (n>1) tasks one of which is chosen and executed, completed when the task completes.
AND_AND	Containing n (n>1) tasks which are executed concurrently, completed when the n tasks complete.
AND_DISC	Containing n (n>1) tasks which are executed concurrently, completed when m out of n (m<n) tasks complete.
OR_OR	Containing n (n>1) tasks s (n>s>1) of which are executed concurrently, completed when the s tasks complete.
OR_DISC	Containing n (n>1) tasks s (n>s>1) of which are executed concurrently, completed when t out of s (t<s) the tasks complete.

Definition 3. Task set (*T*). *T* represents the set of tasks composing the composite service process.

Definition 4. Contain (*C*). *C* represents the relationship between composition patterns and the nodes it contains, and $C = \{(cp, node) \mid cp \in CP, node \in CP \cup T\}$. And

$(cp, node) \in C$, iff $cp \in CP$ and cp contains $node$。 Any node is contained by one and only one pattern, that is to say, if $\exists (p_1, node) \in C$ and $\exists (p2, node) \in C$, then $p_1 = p_2$.

Definition 5. QoS description of tasks (Q). Q describes the QoS value of tasks, and $Q = \{(t, d, v) \mid t \in T\}$, in which $t \in T$, and d represents the type of QoS such as *performance, cost, availability* and *reputation* [1], and v represents the average value of QoS.

3 Transformation

The semantic concepts of BPEL are the abstraction of the information described by the BPEL process specification, which are organized as the metamodel of BPEL. And the transformation rules between BPEL metamodel and the prediction model define the mapping from the concepts in BPEL to the concepts in the general model. And with the transformation rules BPEL processes specification can be converted into the general model which is the input of the prediction algorithm.

The metamodel of BPEL in UML [9] is shown in figure 2, which describes the main semantic elements of BPEL and the relationship among them. The semantic elements concerned with the prediction are *process* and *activity*. The detail mapping rules between the elements and the concepts in general model are described as follows.

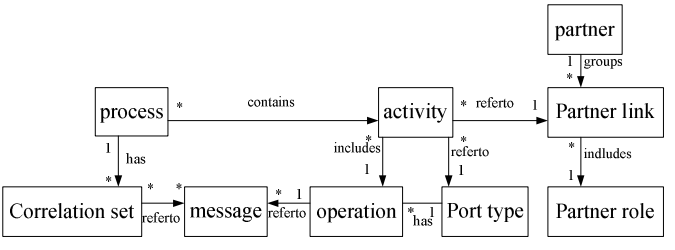

Fig. 2. BPEL metamodel

process element described as <process> in BPEL will be transformed to a composite service process Γ. And *activity* element is split into two types, one is basic activity, the other is structured activity.

3.1 Transformation of Basic Activity

The basic activity in BPEL represents the invocation operation of element services and is described as <invoke>, <receive> and <reply>. Basic activities will be transformed to tasks in composite service process. <invoke> represents a synchronized or asynchronous Web service invocation and will be mapped to a task t. And <receive> and <reply> together represent a synchronized invocation and will be mapped to a task t. The tasks corresponding to all basic activities in <process> make up with the tasks set T.

3.2 Transformation of Structured Activity

The structured activity in BPEL *specifies* the order of activities contained in it, and is described as <sequence>, <switch>, <while>, <pick>, and <flow>. The structured activities will be transformed to composition patterns (*cp*) of composite service process and the relationship between the structured activity and the activities it contains will be transformed to the elements in *C* of the composite service process. The structured activity can be nested and the nested activity can be mapped to nested *cp*.

<sequence> contains one or more activities that are performed sequentially, in the order in which they are listed. It is mapped to a *cp* with *sequence* type. <switch> contains one or more condition branches, one of which will be chosen to be executed according to the condition. It is mapped to a *cp* with *XOR-XOR* type. <while> represents repeated execution of a specified iterative activity until the given Boolean no longer holds true. It is mapped to a *cp* with *Loop* type. <pick> contains one or more event branches, one of which will be chosen to be executed according to the event. It is mapped to a *cp* with *XOR-XOR* type. And <flow> specifies one or more activities to be performed concurrently. It is mapped to a *cp* with *AND-AND* type. All the *cp*s corresponding to the structure activities in <process> make up the composition pattern set *CP*.

3.3 To Get the QoS Description *Q*

QoS description of tasks can not be extracted directly from the BPEL process specification. The QoS of tasks is the QoS of basic activities which is decided by the element service being invoked. There are two ways to get the QoS of services. One is to refer to the interface specification of services which can be located through the *porttype* and *operation* attributes of basic activities. A method of describing service QoS in WSDL specifications has been proposed in [10]. And the other is to get the QoS of services through test or monitoring. A method of getting service QoS through monitoring has been proposed in [11].

4 Prediction Algorithm

Based on the prediction model in section 2 we define the prediction algorithm to predict the average value of the QoS of composite service processes. The QoS attributes supported by the algorithm include *performance, cost* and *reputation*. The algorithm is independent of the specific definition of attributes and can be used for each attribute. In the algorithm, the QoS of processes are based on the QoS of composition patterns calculated through the QoS aggregation formula for composition patterns which take the QoS of the tasks in the pattern as input, and recursive procedures are used for the calculation of nested patterns. And the QoS of the most parent pattern is the prediction result of the process.

For each type of composition patterns and for each attribute of QoS, a QoS aggregation formula is defined. For example, the *performance* QoS aggregation formula for *sequence* composition pattern is defined as $\sum_i x_i$, and for *XOR-XOR*

pattern is $\sum_i p_i x_i$, in which, x_i represents the *performance* value of tasks in the pattern and p_i represents the probability of the branch including the task being executed. The formulas for each pattern type and for the attributes of *performance, cost and reputation* are given in [6].

4.1 Algorithm Definition

The algorithm inputs include the composite service process Γ and the attribute *d* to be predicted. Through the algorithm the average value of the *d* attribute of the process Γ can be computed.

The algorithm is made up of three steps. First, the process is composed of nested composition patterns and we need to get the most parent composition pattern described as cp_0 and satisfying $\forall cp \in CP, (cp, cp_0) \notin C$. Second, in order to calculate the QoS of composite patterns, we need to get the attribute value of *d* of all nodes in cp_0. And then let *node* represent a node in cp_0. When *node* is a task, the QoS of *node* is the *v* in $(node, d, v) \in D$. And when *node* is a nested composition pattern, the QoS of *node* can be extracted through the recursive invocation of the algorithm. Thirdly, take the QoS value of nodes in cp_0 into the aggregation formula for the attribute of *d* and the pattern type of cp_0 to get the QoS of the composition pattern, which is the result of prediction.

4.2 Time Complexity

Theory 3.1**.** As the input is $\Gamma = (CP, T, C, Q)$, the time complexity of the algorithm is $O(|CP|) + O(|T|)$.

Let $|CP| = n$, $|T| = m$, and there is $|C| = m + (n-1)$. To get the most parent composition pattern, the algorithm checks each element in *C* and debars those being contained by other patterns. The worst time complexity of the procedure is $(m + n - 1) \cdot t_1$, in which t_1 represents the constant time to deal with a single element in *C*. In addition, the time of the procedure of getting the QoS of all tasks is $m \cdot t_2$ and of getting the QoS of all composition patterns is $n \cdot t_3$, in which t_2 and t_3 respectively represent the constant time of getting the QoS of a service and of calculating an aggregation formula. As a result, the time performance of the algorithm is $(m + n - 1) \cdot t_1 + m \cdot t_2 + n \cdot t_3$, and the complexity is $O(n) + O(m)$, that is $O(|CP|) + O(|T|)$.

5 Experiment

To prove the feasibility and accuracy of the approach proposed in the paper, we give an example of BPEL process prediction. First, the performance of a BPEL process is predicted by the approach, and then the BPEL process is executed by Active BPEL to

get the actual average process performance, and the actual value and prediction result are compared.

The example BPEL process specification is shown in figure 3, and the function of the process is to perform stock quote or exchange rate consultant according to the user request. There are four element services: *Receipt*, *StockQuote*, *ExchangeQuote* and *Response*, and the WSDL file URLs of the element services are shown in table 2.

And the definition of Web services *performance* is given in [1] as the interval between receiving the request and sending out the response, and the interval can be measured through the execution time of services.

```
<Process name=”ConsultantService” …>
<partnerLinks>…</partnerLinks>
<partners>…</partners>
<variables>…</variables>
<sequence>
    <invoke partnerLink="Receipt"
        portType="urn:ReceiptService" operation="ConsultantRequestReceipt"
        variable="ConsultantRequest"  createInstance="yes"…/>
    <switch>
        <case condition="urn:RequestServiceType==1">
            <invoke partnerLink="StockQuote"
                portType="urn:StockQuoteService" operation="StockQuoting"…/>
        </case>
        <case condition=" urn:RequestServiceType==0">
            <invoke partnerLink="ExchangeQuote"
                portType="ExchangeQuoteService" operation="ExchangeQuoting"…/>
        </case>
    </switch>
    <invoke partnerLink="Response"
        portType="urn:ResponseService" operation="ConsultantResponser"
        variable="ConsultantResponse" createInstance="yes"…/>
</sequence>
</Process>
```

Fig. 3. The example BPEL process

Table 2. The WSDL file URLs of the element services in the example BPEL process

element service	WSDL URL
Receipt	http://www.telestar.bj.cn/spacejojo/wsdl/receipt.wsdl
StockQuote	http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
ExchangeQuote	http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl
Response	http://www.telestar.bj.cn/spacejojo/wsdl/response.wsdl

The performance value of the example process can be computed as follow. The BPEL process is transformed to a corresponding prediction model according to the rules in section 4 to get the input of the algorithm, and the result prediction model is shown in table 3. And the performance of element services represented as the QoS description Q in the prediction model is gotten through service test. We record the performance data of element services in ten executions and take the average

performance as the value in Q. And we suppose the branches in <switch> have the same probability to be executed. Through the prediction algorithm, we can get the prediction result of the process performance as16.05*s*.

Table 3. The result prediction model transformed from the example BPEL process

Element	Value
Γ	(CP, T, C, Q)
T	{*t_receipt*, *t_stock*, *t_exchange*, *t_response*}
CP	{*cp_sequence*, *cp_switch*}
C	{(*cp_sequence*, *t_receipt*), (*cp_sequence*, *cp_switch*), (*cp_sequence*, *t_response*), (*cp_switch*, *t_stock*), (*cp_switch*, *t_exchange*)}
Q	{(*t_receipt*, *performance*, *3.3*), (*t_stock*, *performance*, *10.2*), (*t_exchange*, *performance*, *9.3*), (*t_response*, *performance*, *3.0*)}

Table 4. The run-time performance of the example process and the included basic activities

instance	*Process* <sequence>	Receipt <invoke>	<switch>		Response <invoke>
			StockQuote <invoke>	*ExchangeQuote* <invoke>	
1	16.97	3.67	–	9.32	2.54
2	15.46	3.00	–	9.05	3.13
3	18.67	3.77	–	9.06	3.34
4	17.5	3.68	10.16	–	3.33
5	18.02	3.78	10.55	–	3.42
6	19.63	3.06	10.53	–	3.47
7	19.09	3.64	10.00	–	3.47
8	15.72	2.99	–	9.30	3.04
9	16.53	3.41	–	9.78	2.99
10	16.3	3.67	–	8.95	2.91
11	16.84	3.08	10.05	–	2.89
12	16.11	3.63	–	9.79	3.35
13	18.06	3.08	10.36	–	2.96
14	16.05	3.40	–	9.31	3.31
15	17.73	3.35	10.44	–	3.36
16	15.42	3.20	10.04	–	3.47
17	16.53	3.57	–	9.68	3.11
18	16.81	2.97	10.01	–	3.33
19	15.39	2.93	–	9.26	2.89
20	16.81	3.54	10.43	–	2.91
Avg	16.98	3.36	10.24	9.35	3.18

And then the example process is executed by Active BPEL, which is an execution engine of BPEL based on Java [13]. The reason to choose Active BPEL is that it provides the log on the start and end time of the process as well as the basic activities in the process, so that the run-time performance of the process and the basic activities can be gotten. Active BPEL takes the process specification and the element service WSDL file URL as the input. According to the Active BPEL logs on 20 executions of the example process, the run-time performance of the process and the basic activities can be calculated and the result is shown in tabel 4. The actual average process performance can be gotten as 16.98*s*.

From the data in table 3 and table 4, we can see that there are errors between the estimation and the actual performance of element services, and this is one of the reasons leading to the error of process prediction. The process prediction error rate can be calculated through the formula of $\eta = \frac{|V_e - V_a|}{V_a}$ in which V_e and V_a respectively represent the prediction result and actual value. And the error rate of the performance prediction of the example process is 5.5%, which proves the approach has a good estimation in predicting the average QoS value of BPEL processes.

6 Conclusion

A model-driven approach for the QoS prediction of BPEL process is proposed in the paper. It has a two-layer architecture: One is the prediction model specifying necessary information on composite services for prediction and independent of specific languages, and the other is the semantic model of specific languages such as BPEL. A set of transformation rules is defined between the two layers so that processes specified in specific languages can be transformed to the prediction model. Based on the prediction model the prediction algorithm to predict the average value of the QoS attributes of composite service processes is defined. And we've proved the algorithm has a linear time complexity. Model-driven strategy makes it possible for the approach to be used for the prediction of processes specified in any languages, if the sets of mapping rules between the semantic models of the language and the prediction model are defined. The feasibility and good accuracy of the approach in BPEL process prediction has been proved by the experiment.

Acknowledgment. We thank the National Basic Research Priorities Programme (Grant No. 2003CB314806) for funding the project.

References

1. Zeng, L., Benatallah, B., Ngu, A.H.H., et al.: QoS-Aware Middleware for Web Services Composition. Software Engineering, IEEE Transactions on 30(5), 311–327 (2004)
2. Grassi, V.: Architecture-based Reliability Prediction for Service-oriented Computing. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems III. LNCS, vol. 3549, Springer, Heidelberg (2005)

3. Chadrasekaran, S., Miller, J.A., Silver, G.S., et al.: Composition, performance analysis and simulation of web services. Electronic Markets: The International Journal of Electronic Commerce and Business Media (2003)
4. Cardoso, J.: Quality of Service and Semantic Composition of Workflows. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA) (2002)
5. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS aggregation for service composition using workflow patterns. In: EDOC 2004. Proceedings of the 8th International Enterprise Distributed Object Computing Conference, Monterey, California, IEEE Computer Society Press, Los Alamitos (2004)
6. Jaeger, M.C., Rojec-Goldmann, G., Muhl, G.: QoS aggregation in Web service compositions. In: EEE 2005. Proceedings of the IEEE Int. Conf. on e-Technology, e-Commerce and e-Service, pp. 181–185. IEEE Computer Society Press, Los Alamitos (2005)
7. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1, OMG (2003)
8. van der Aalst, W.M.P., van Hee, K.M., Houben, G.J.: Modeling workflow management systems with high-level Petri nets. In: Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pp. 31–50 (1994)
9. Mongiello, M., Castelluccia, D.: Modelling and Verification of BPEL Business Processes. In: MBD/MOMPES 2006. Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (2006)
10. Gouscos, D., Kalikakis, M., Georgiadis, P.: An Approach to Modeling Web Service QoS and Provision Price. In: Proceedings of the Fourth International Conference on Web Information Systems Engineering, pp. 121–130 (2003)
11. Liu, Y., Ngu, A.H.H., Zeng, L.: QoS Computation and Policing in Dynamic Web Service Selection. In: Proceedings of the Thirteenth International World Wide Web Conference, New York (2004)
12. van der Aalst, W.M.P.: Web Service Composition Languages: Old Wine in New Bottles. In: Proceedings of the 29th EUROMICRO Conference New Waves in System Architecture EUROMICRO (2003)
13. Stoilova, K., Stoilov, T.: Comparison of workflow software products. In: CompSysTech. Proceedings of the International Conference on Computer Systems and Technologies (2006)