World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:1, 2007

Adaptive Algorithm to Predict the QoS of Web Processes and Workflows

Jorge Cardoso

***Abstract*—**Workflow Management Systems (WfMS) allow organizations to streamline and automate business processes and reengineer their structure. One important requirement for this type of system is the management and computation of the Quality of Service (QoS) of processes and workflows. Currently, a range of Web processes and workflow languages exist. Each language can be characterized by the set of patterns they support. Developing and implementing a suitable and generic algorithm to compute the QoS of processes that have been designed using different languages is a difficult task. This is because some patterns are specific to particular process languages and new patterns may be introduced in future versions of a language. In this paper, we describe an adaptive algorithm implemented to cope with these two problems. The algorithm is called adaptive since it can be dynamically changed as the patterns of a process language also change.

***Keywords*—**Quality of Service, Web processes, Workflows, Web services.

I. Introduction

With the mergence of Web services, workflow management systems have become essential to support, manage, and enact workflows and Web processes, both between enterprises and within the enterprise [1].

The management of the QoS of Web processes is an important requirement for organizations operating in modern markets involving e-commerce and e-business activities using distributed Web services interactions.

The implementation of QoS management support involves the development of integrated solutions composed of four modules: specification, prediction, monitoring, and control [2]. In this paper, we turn our attention to the development of prediction algorithms. Prediction algorithms are important since they allow the computation of a Web process QoS before making the process available to its customers.

Currently there is no agreement on the patterns or control-flow pattern that should be part of Web processes or workflows languages [3]. Therefore, prediction algorithms need to be tailored to fit the patterns of specific process specification languages. As a result, slightly different implementations of the same prediction algorithm to compute the QoS of processes need to be developed for each process specification language.

It is highly desirable and important to develop prediction algorithms that could be easily adapted to reflect the patterns present in a process specification language. Adaptable prediction algorithms for QoS computation can be dynamically changed to cope with the introduction of new patterns. The development of such algorithms is the main objective of this paper.

This paper is structured as follows. In Section 2, we discuss the basis of algorithms to compute and predict the QoS of processes. Namely, we present a static algorithm which can be used to compute and predict the QoS of processes, the SWR algorithm, and discuss the inherent need for adaptive algorithms. Section 3 discusses the characteristics of static and adaptive QoS prediction algorithms. We present an adaptive version of the SWR algorithm which is more suitable for current WfMSs. Finally, section 4 presents our conclusions.

II. QoS prediction algorithms

The design and composition of processes cannot be undertaken while ignoring the importance of QoS measurements. The management of QoS directly impacts the success of organizations participating in e-commerce.

One important requirement of algorithms to compute the QoS of Web processes is the ability to automatically compute the QoS of Web processes based on the QoS of process components (i.e., Web services). This feature is important, especially for large processes that in some cases may contain hundreds of Web services. A manual computation is neither realistic nor viable.

A. Describing the structure of a process

Web process, workflows and processes in general can be specified using specification languages such as BPEL4WS and WSFL. A Web process schema is the actual topology of a process, that is, the sequence of Web services which must be performed in order to accomplish an organizational goal. Web process and processes in general can be specified using a specification language which include a set of fundamental patterns. An example of a Web process is illustrated in Fig. 1.

Manuscript received October 29, 2004.

J. Cardoso is with the Department of Mathematics and Engineering, University of Madeira, Funchal, 9000-019 Portugal, (e-mail: jcardoso@uma.pt).

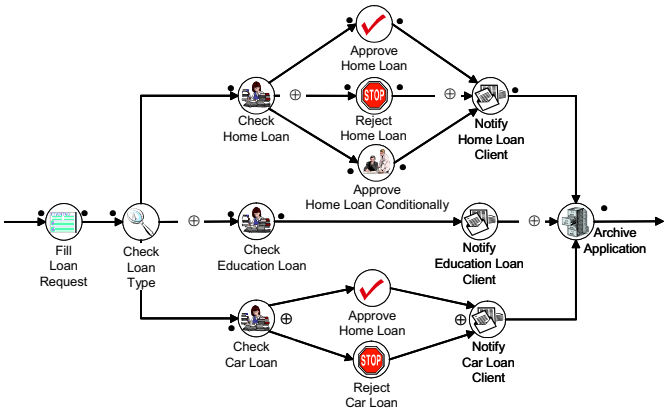

FIGURE I. Example of a Web process

A Web service with more than one outgoing transition can be classified as an AND-split or XOR-split. AND-split Web services enable all their outgoing transitions after completing their execution. AND-split Web services enable only one outgoing transition after completing their execution. And-split Web services are represented with a '•' and XOR-split Web services are represented with a '⊕'. A Web service with more than one incoming transition can be classified as an and-join or XOR-join. AND-join Web services start their execution when all their incoming transitions are enabled. XOR-join Web services are executed as soon as one of the incoming transitions is enabled. As with AND-split and XOR-split Web services, AND-join Web services and XOR-join Web services are represented with the symbol '•' and '⊕', respectively.

B. Computing Web processes' QoS

We identify two methods which can be used to compute the overall QoS of processes: mathematical modeling and simulation. Simulation will not be discussed in this paper and the reader is referred to [4] for the description of simulation analysis techniques and systems.

Mathematical modeling methods formally describe the formulae to compute QoS metrics among Web services. In the next section, we present an algorithm that we have developed to automatically compute the overall QoS of a process.

C. The SWR algorithm

To compute QoS metrics for Web processes based on Web service's QoS metrics the Stochastic Workflow Reduction (SWR) algorithm [5] can be used. While the algorithm we have developed was initially implemented for workflows, it can also be applied to Web processes.

The SWR algorithm repeatedly applies a set of reduction rules to a Web process until only one Web service remains. Each time a reduction rule is applied, the process structure changes. After several iterations only one Web service will remain. When this state is reached, the remaining Web service contains the QoS metrics corresponding to the Web process under analysis.

The algorithm uses a predefined set of six reduction rules because a vast majority of workflow systems support them [3]. The algorithm has been designed for a specific set of patterns that are part of the METEOR [6] workflow language. The algorithm and the rules are hard-coded and cannot be changed easily. We will see in the next section that this is a limitation that should be overcome.

D. The need for an adaptive algorithm

A large number of process specification languages exist nowadays. While some of these languages have existed for more than a decade now, there is a lack of consensus as to what constitutes a Web process specification. A modest agreement has been reached as to what should be key components of a process specification language [3].

Process specification languages, such as workflow languages, can de described based on the control flow constructs that they allow to be modeled. These constructs have been denominated patterns [7]. Patterns describe control flow dependencies which can be modeled using a particular process modeling language and address business requirements.

It is important to realize that patterns are derived from business requirements. Depending on the application domain it may be essential to create additional patterns to represent and handle particular types of control flow structures. When new patterns are needed to design processes that require a specific set of requirements, the QoS of the newly created process can no longer be computed and analyzed using a static algorithm. Computation is not feasible since new patterns and rules govern the process specification. One obvious solution is to rewrite and recompile the algorithm to cope with the newly introduced patterns. Another more attractive solution would be to use an adaptive algorithm which could accept new patterns as they are needed for a particular application domain.

In the next section we demonstrate how such an adaptive algorithm to compute the QoS of Web processes can be designed and implemented.

III. From a Static to an Adaptive QoS Prediction Algorithm

In this section we present two versions of the SWR algorithm. The first version is a static version of the algorithm. Any modification or adaptation to support new process patterns requires recoding and recompiling the algorithm. The second version of the algorithm is adaptive in the sense that new patterns can be dynamically added or retracted without the need to recompile or rebuild the algorithm.

A. SWR algorithm: The static version

Comprehensive solutions to the difficult problems encountered in synthesizing QoS for composite Web services (Web processes) are discussed in detail [8], which presents the SWR algorithm. This algorithm is able to compute metrics such as the response time (T), cost (C) and reliability (R).

The SWR algorithm has six reduction rules (sequential, parallel, conditional, fault-tolerant, loop, and network) hard-coded and cannot be changed easily, they are static and so is

the algorithm.

As an illustration, we will show how reduction works for a parallel system. Fig. II shows how a system of parallel Web services t_1, t_2, …, t_n, an AND-split Web service t_a, and an AND-join Web service t_b can be reduced to a sequence of three Web services t_a, t_{1n}, and t_b. In this reduction, the incoming transitions of t_a and the outgoing transition of Web services t_b remain the same. The only outgoing transitions from Web service t_a and the only incoming transitions from Web service t_b are the ones shown in the figure below.

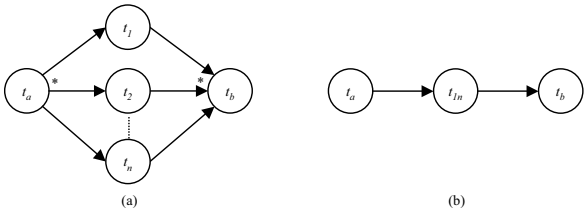

FIGURE II. Parallel system reduction

The QoS of Web services t_a and t_b remain unchanged. To compute the QoS of the reduction the formulae in Table I are applied.

TABLE I
REDUCTION FORMULAE TO COMPUTE QOS

$$T(t_{1n}) = Max_{i\in\{1..n\}}\{T(t_i)\} \qquad C(t_{1n}) = \sum_{1\le i\le n} C(t_i) \qquad R(t_{1n}) = \prod_{1\le i\le n} R(t_i)$$

The SWR algorithm has been coded in Java and can be found in [5].

3. *SWR algorithm: The adaptive version*

One of the major requirements for the development of an adaptive algorithm is its ability to dynamically accept new patterns and compute the QoS of processes with previously unseen control flow structures.

One solution to develop such an algorithm is to use predicate logic. We choose predicate logic to achieve five major goals: a) formally describe the structure of a process, b) reduction rules definition, c) pattern identification, d) QoS model computation, and e) pattern reduction. Each of these goal are discussed and illustrated in the following sections.

1) Process structure specification

To represent a Web process, such as the one in Fig. 1, each of the components of the process schema need to be translated to a suitable presentation in predicate logic. We give details about the facts that describe a process schema and the QoS model associated with each Web service. We represent in predicate logic the following basic components of a process: Web services, start service, end service, transitions, XOR-splits, XOR-joins, AND-splits, and AND-join.

Web service. We use the predicate `webservice/1` to represent a Web service. The following examples are extracted from the Web process illustrated in Fig. 1.

```
webservice( fill_loan_request ).
webservice( check_loan_type ).
webservice( check_home_loan ).
webservice( archive_application ).
```

Start service and end service. The predicates for these two elements are `start/1` and `end/1`. For example,

```
start(start_service).
end(end_service).
```

Transitions. Transitions are directed arcs that express dependencies between Web services. For transitions the predicate is `transition/3`. The first argument indicated the probability p of the transition being fired at runtime, the second argument indicates the Web service source and the last argument represents the Web service target. For example,

```
transition(  1, startservice,    fill_loan_request).
transition(0.4, check_loan_type, check_home_loan).
transition(0.3, check_loan_type, check_car_loan).
```

XOR-splits, AND-splits, XOR-joins, AND-joins. These elements are used to capture the execution logic in processes and are represented with the predicates `xorsplit/1`, `xorjoin/1`, `andsplit/1`, and `andjoin/1`, respectively. From our example the following facts hold,

```
xorsplit(check_loan_type).
xorjoin(archive_application).
```

QoS model. Process QoS addresses the non-functional issues of processes and can be characterized along various dimensions. The QoS model employed in this paper is composed of three dimensions: *time*, *cost*, and *reliability*. A complete description of the model can be found in [2].

TABLE II
EXAMPLE OF QOS SPECIFICATION FOR A WEB SERVICE *FILL FORM* [2]

	Min value	Avg value	Max value
Time	192	196	199
Cost	576	576	576
Reliability	-	100%	-

The basic class of the QoS model of the *Fill Form* Web service from Table II is translated to the following first order logic statement:

```
qos(fillform, time(192,196,199), cost(576,576,576),
    reliability(0,1,0)).
```

2) Rule Definition

The following section of code illustrates the main steps involved when applying any reduction rule. Each rule is composed of three main segments: a) pattern identification, b) QoS computation and c) pattern reduction. For a sequential reduction rule the segments are the following:

```
applySequentialRule(SrcWebservice) :-
    a) isaSequentialStructure(SrcWebservice,
       DstWebservice),
    b) computeQoSSequentialSystem(SrcWebservice,
       DstWebservice, QoSModel),
```

```
c) reduceSequentialSystem(SrcWebservice,
   DstWebservice, QoSModel).
```

Each segment is discussed individually in the next sections.

3) Pattern Identification

All the patterns that compose a process language must be translated into a first order logic form. We have translated the six patterns for which a reduction rule has been developed for the SWR algorithm.

Due to space limitation, we will only describe the sequential pattern reduction rule. The first step is to identify if a particular process pattern exists in a process. Each individual pattern is recognized based on the intrinsic characteristic of the patterns themselves.

```
isaSequentialStructure(SrcWebservice, DstWebservice) :-
  \+ network(SrcWebservice),
  \+ network(DstWebservice),
  \+ xorsplit(SrcWebservice),
  \+ xorjoin(SrcWebservice),
  \+ andsplit(SrcWebservice),
  \+ andjoin(SrcWebservice),
  \+ xorsplit(DstWebservice),
  \+ xorjoin(DstWebservice),
  \+ andsplit(DstWebservice),
  \+ andjoin(DstWebservice),
  \+ start(SrcWebservice),
  \+ end(DstWebservice),

  getNextWebservices(SrcWebservice, NextWebservices),
  length(NextWebservices, 1),
  member(DstWebservice, NextWebservices),

  getPrevWebservices(DstWebservice, PrevWebservice),
  length(PrevWebservices, 1),
  member(SrcWebservice, PrevWebservices).
```

Check if the Web services involved are not networks, splits or joins, and start or end services.

Check if the Web service source has only one outgoing transition

Check if the next Web service has only one incoming transition

4) QoS Model Computation

When it has been determined that a set of Web services form a specific pattern, the computation of its QoS can be evaluated. For example, if two Web services constitute a sequential system, then the QoS of the sequential system is computed the following way.

```
computeQoSSequentialSystem(SrcWebservice,
                           DstWebservice, NewQoSModel) :-
  a) qos(SrcWebservice, _Ts, _Cs, _Rs),
  b) qos(DstWebservice, _Td, _Cd, _Rd),
  c) addQoSmodels(qos(SrcWebservice, _Ts, _Cs, _Rs),
     qos(DstWebservice, _Td, _Cd, _Rd), NewQoSModel).
```

In line a) and b) the QoS model of each Web services is obtained and in line c) the two QoS models are added resulting a new QoS model. While we do not give specific details on the computation of QoS models, the reader may refer to [2] to find additional information.

5) Pattern Reduction

Once the QoS of a pattern has been computed, the pattern needs to be replaced with an equivalent pattern from the QoS point of view. This replacement is illustrated in Fig. II, where the pattern a) can be replaced with pattern b) after calculating the QoS [2].

In our running example, the reduction of such a system involves removing existing transitions (a), Web services (c,d), and QoS models (e,f). The reduction also involves, updating transitions (h) and adding a new Web service and its QoS model (g,i). These actions are illustrated in the following segment of code.

```
reduceSequentialSystem(SrcWebservice, DstWebservice,
                                          QoSModel) :-
  a) retract(transition( _p, SrcWebservice,
     DstWebservice)),
  b) QoSModel = qos(NewWebservice,_,_,_),
  c) retract(webservice(SrcWebservice)),
  d) retract(webservice(DstWebservice)),
  e) retract(qos(SrcWebservice, _,_,_)),
  f) retract(qos(DstWebservice, _,_,_)),
  g) assert(webservice(NewWebservice)),
  h) updateTransitions(SrcWebservice,
     DstWebservice, NewWebservice),
  i) assert(QoSModel).
```

C. Adding and removing pattern

In order to add new rule definitions to compute of the QoS of previously unknown workflow patterns, it is only necessary to implement the five main elements described in the previous sections. Once these elements have been implemented they can be dynamically introduced into the algorithm without requiring any other changes. To remove a pattern that is no longer supported by a process language is simple since it is only necessary to retract the rules definition of the pattern.

IV. Conclusions

Developing a suitable algorithm to compute the Quality of Service of Web processes designed using several process languages is a difficult task. This is because some patterns are specific to particular languages and a few patterns may have been introduced after implementing the algorithm.

In this paper we show an adaptive algorithm implemented using a first order logic programming language. The algorithm is able to cope with various known process patterns and new ones as they are deployed. The algorithm is called adaptive since it can be dynamically changed as the patterns of a process language also change.

References

[1] Sheth, A.P., W.v.d. Aalst, and I.B. Arpinar, Processes Driving the Networked Economy. IEEE Concurrency, 1999. 7(3): p. 18-31.

[2] Cardoso, J., et al., Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web Journal, 2004. 1(3): p. 281-308.

[3] Aalst, W.M.P.v.d., et al. Advanced Workflow Patterns. 7th IFCIS International Conference on Cooperative Information Systems. 2000.

[4] Miller, J.A., J.S. Cardoso, and G. Silver. Using Simulation to Facilitate Effective Workflow Adaptation. in Proceedings of the 35th Annual Simulation Symposium (ANSS'02). 2002. San Diego, California.

[5] Cardoso, J., Stochastic Workflow Reduction Algorithm. 2002, LSDIS Lab, Department of Computer Science, University of Georgia.

[6] Kochut, K.J., METEOR Model version 3. 1999, Large Scale Distributed Information Systems Lab, Department of Computer Science, University of Georgia: Athens, GA.

[7] Aalst, W.M.P.v.d., et al., Workflow Patterns. 2000, Eindhoven University of Technology: Eindhoven.

[8] Cardoso, J., A. Sheth, and J. Miller. Workflow Quality of Service. in International Conference on Enterprise Integration and Modeling Technology and International Enterprise Modeling Conference (ICEIMT/IEMC'02). 2002. Valencia, Spain: Kluwer Publishers.