Monitoring Web Services: A Database Approach*

Mohamed Amine Baazizi[1], Samir Sebahi[1], Mohand-Said Hacid[1], Salima Benbernou[1], and Mike Papazoglou[2]

[1] University Claude Bernard Lyon 1, LIRIS CNRS UMR 5205, France
[2] Tilburg University, The Netherlands

Abstract. Monitoring web services allows to analyze and verify some desired properties that services should exhibit. Such properties can be revealed by analyzing the execution of the services. Specifying monitoring expressions and extracting relevant information to perform monitoring is however not an easy task when the processes are specified by means of BPEL. In this paper we design a monitoring approach that makes use of business protocols as an abstraction of business processes specified by means of BPEL. High level queries are expressed against this abstraction and then translated into SQL queries that are evaluated against a database that stores the excustion traces of the services.

1 Introduction

The task of observing some process and tracking specific situations is known as monitoring. Amongst the many fields that witness special need of monitoring, there are Service Based Systems (SBS) which tend to support most today's applications. Today's enterprises rely on SBS to export their products. Different stakeholders may communicate with each other combining their existing products to constitute other ones more tangible to end users. This interaction is specified in a complex manner and advocates defining all the activities taking place in it. Moreover, each participant operates in an information system different from those used by the other parties. This was overcome by establishing a stack of protocols leveraging the heterogeneity that could take place. However, planing monitoring for those systems remains difficult since it advocates not only knowing details of the process being monitored, but also mastering tools and languages that served to specify it. All this to say that a new way to monitor business processes is more than required. It should take into account the difficulties of finding suitable models that bridge the gap between the modeled processes and the restricted knowledge of decision making actors.

In this paper, we build on previous work by Wombacher *et al.* [5] and Benatallah *et al.* [4] to provide a methodology for monitoring web services by considering their business protocols. The methodology is shown on figure 1. A BPEL specification is transformed into a business protocol. From the business protocol we

* The research leading to this result has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-CUBE).

P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, LNCS 5377, pp. 98–109, 2008.

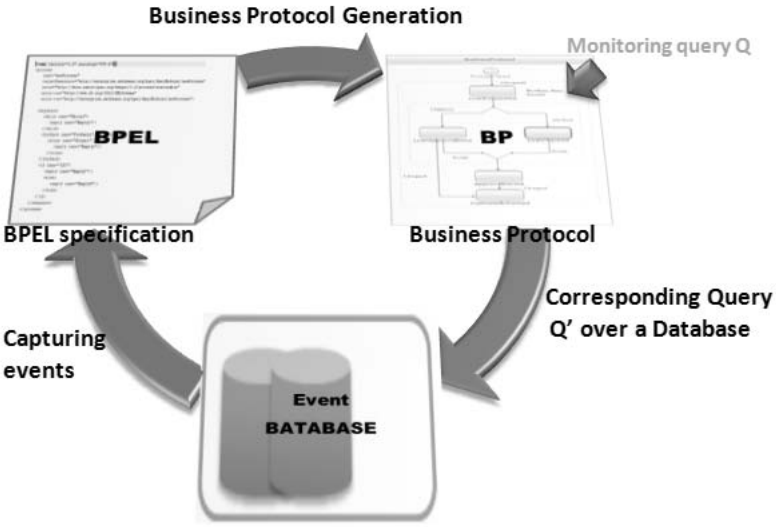

Fig. 1. The monitoring Framework

generate a (relational) database schema. Monitoring queries are specified against the protocols and translated into SQL queries against the relational database.

The paper is organized as follows: we first introduce, in section 2, a few notions on web services that are relevant in our work. Section 3 defines the monitoring process in web services. Section 4 describes our architectural and design principles of our approach for monitoring web services. We conclude in section 5 by summarizing our work and anticipating on necessary extensions.

2 Preliminaries

In this section we introduce relevant concepts to our work. We assume the reader familiar with Service Oriented Architectures (SOA) and notions related to Web services.

2.1 Business Process Execution Language (BPEL)

A business process consists of a bounded set of activities where data is manipulated and results are produced following the logic of the business it describes. Before the advent of web services, enterprises workflows were broadly used in describing the collaboration of many actors to produce a result. This was modeled by a graphical notation that shows the activities to be performed and the scheduling to respect in addition to the intermediate products that are passed between activities. In the same way, web services collaboration is captured by Business Process Execution Language (BPEL) [1], an XML-based standard used to orchestrate the enactment of the different services of the collaboration that interact

to perform some specified task. It specifies the process behavior by defining the activities it is composed of and the external processes that interact with it.

2.2 Business Protocols

Making several partners' processes collaborate in an effective way needs an a priori look on their descriptions to depict eventual mismatches before their enactment. The actual standard for specifying web service offers neither enough visibility on the processes it specifies nor suitable tools that could help designers statically analyze the behavior of these processes that they manage to make communicate in a correct manner. This is why the authors in [4] investigated a way to represent the protocol that two services must follow to interact correctly. This was called business protocol since it represents the allowed conversations between a requester and a provider in terms of messages according to the states that they reached in the local execution of their respective business processes.

3 Monitoring Web Services

Web services are characterized by the fact that they are contracted somewhere in the time and may not be available after that or can still be available but in a different version making their evolution highly volatile. Additionally, every participant is mandated to correctly perform the task it has to carry out otherwise it will affect the entire process execution.

Monitoring copes with those deficiencies by observing web services execution after they have been deployed. It consists of a dedicated activity responsible of raising alert or triggering predefined actions when some situation is observed. It also consists of gathering useful information that will serve analysis. It could be extended with capabilities that allow avoiding some unwanted situations.

Monitoring web services was influenced by many techniques dealing with contracts and agreements, distributed systems property and safety verification, event processing, etc.

Many criteria could be considered when classifying monitoring approaches. According to [2], we can focus on the technique used to perform monitoring (verification, planning...) as well as on the data of interest to be monitored and many other aspects such as the abstraction of the language that serves monitoring specification and the degree of invasiveness on the monitored process.

4 A Database Approach for Monitoring Web Services

In this section we define the framework we are designing for monitoring business processes specified in BPEL. We will detail each component's functionality and the transformations undergone. We also provide the language used for formulating monitoring queries and characterize them regarding the abstraction upon which they are expressed.

4.1 The Overall Architecture

Figure 1 depicts the main components of the framework and the transformations that lead to each of them. We consider the executable BPEL specification of the business process to monitor. This specification will be mapped to a corresponding business protocol, provided some changes that will be discussed later. The mapping operation rests on a set of required transformation rules. A query language is then used for retrieving information by navigating through the states of the business process. Each query will be transformed into a suitable SQL query over a database which schema is a faithful mapping of the business protocol resulting from the transformation of the business process. This database is populated during the execution of the service supporting the business process.

4.2 A Business Protocol as an Abstraction

The abstraction of BPEL that we consider is a business protocols defined in [4] that we extend with variables associated with the states. The core definitions are kept identical. A protocol is defined as a tuple A=(Q, q_0, F, ϕ , Σ ,ψ, Var) where:

- Q is a finite set of states the process goes through during its execution
- q_0 is the initial state
- F$\subseteq$ Q is the set of final states where F$\neq \emptyset$
- ϕ is the set of messages. There are two types of messages, those consumed by the protocol, these are assigned the polarity sign + and those produced by the protocol are assigned the - sign.
- $\Sigma \subseteq Q \times \phi \times Q$ is the transition set where every transition is labeled with a message name and its polarity.
- ψ is a partial function that assigns to the states where a transition labeled with a receive message enters, the variable that is modified by this message. Not all states are assigned variables since only entering messages deliver information that is recorded in their corresponding variables.
- Var is the finite set of variables of the business process to be transformed.

4.3 Transformation of BPEL Business Processes to Business Protocols

In this section, we are interested in the mechanism that allows to generate an abstraction of a business process specified in BPEL by a set of rules. First, we have to define the different elements of a BPEL specification as stated in its specification [1].

For transformation purpose, we proceed by generating segments of the protocol corresponding to the basic activities and then combine the resulting segments by looking into the structured activities to which they belong.

4.3.1 Transformation of Basic Activities

For each activity represented in BPEL syntax, we give its corresponding protocol segment definition. States named as q indexed with an integer i are just used for representation and could be renamed.

invoke activity
<**INVOKE** PARTNERLINK="PL" PORTTYPE="PT" OPERATION="OP" INPUTVARIABLE="INVAR" OUTPUTVARIABLE="OUTVAR"/>

is mapped into the following segment of the protocol

$(\{q_i, q_{i+1}, q_{i+2}\}, q_i, \{q_{i+2}\}, \{\mathrm{m}, \mathrm{n}\}, \{(q_i, (-)m, q_{i+1}), (q_{i+1}, (+)n, q_{i+2})\}, \psi(q_{i+1}) = inVar, \psi(q_{i+2}) = outVar, \{inVar, outVar\})$

Here, q_{i+1} is an intermediate state meaning that a message has been sent from a process to one of its partners and is blocked waiting for a message to be returned to change its state and affects the variable defined in this state.

receive activity
The receive activity which waits for a message that will be consumed takes the form <**RECEIVE** PARTNERLINK="PL" PORTTYPE="PT" OPERATION="OP" VARIABLE="VAR">

and is mapped to the protocol defined by

$(\{q_i, q_{i+1}\}, q_i, \{q_{i+1}\}, \{\mathrm{n}\}, \{(q_i, (+)n, q_{i+1})\}, \psi(q_{i+1}) = Var, \{Var\})$

assign activity
<**ASSIGN**><COPY> <FROM>...</FROM> <TO VARIABLE="VAR".../> </COPY> <ASSIGN/> is mapped to its corresponding protocol

$(\{q_i, q_{i+1}\}, q_i, \{q_{i+1}\}, \{\}, (q_i, Assign, q_{i+1}), \psi(q_{i+1}) = Var, \{Var\})$

The assign activity is local to a process and does not require any message exchange. This is why no polarity sign is used.

reply activity
<**REPLY** PARTNERLINK="PL" PORTTYPE="PT" OPERATION="OP" VARIABLE="VAR"> is mapped to the protocol

$(\{q_i, q_{i+1}\}, q_i, \{q_{i+1}\}, \{\mathrm{m}\}, \{(q_i, (-)\mathrm{m}, \mathrm{q_{i+1}})\}, \psi(\mathrm{q_{i+1}}) = \mathrm{Var}, \{\mathrm{Var}\})$

Other activities like wait, exit, empty, throw and rethrow are available in the BPEL specification but not all are relevant. Wait which makes the process wait for a precise moment or until a certain time could be mapped to a business protocol using temporal transitions defined in [3] that are implicit transitions to be taken when the time constraint defined for them is satisfied. Exit is mapped to a transition leading to a final state.

4.3.2 Transformation of Structured Activities

Structured activities are used to link between basic activities following a logic we have in mind at design-time. This is done using different constructs such as **flow** which expresses that the activities defined in its scope run concurrently, **sequence** which links between basic or structured activities that are designed to run sequentially, **if-then-else** express conditional branching to a point in the

process, **while** and **repeat-until** are used to loop through a set of activities and **pick** waits for a suitable message to trigger the corresponding action or a default action if time overruns. As done for the basic activities, we assign for each type of activity given in BPEL syntax its corresponding automaton definition.

4.4 A Monitoring Query Language

The monitoring methodology we propose consists of querying the business protocol corresponding to the business process we want to monitor rather than handling this latter itself. This is why we define our monitoring language upon business protocols to take advantage of the abstraction they offer. A business protocol represents the modeled system as a finite state automaton which transitions are annotated with messages exchanged and states are the mapping of the steps the process goes through until it ends. This visual representation of a system greatly simplifies its comprehension, and could hence be exploited to express queries in a natural and efficient manner. Figure 2 shows the business protocol of loan process system obtained from the transformation of BPEL code provided with the specification [1] using the transformation rules stated in § 4.3. The process starts by receiving customers' requests and decides, based on the asked amount, whether to check the loan request by the assessor service whose

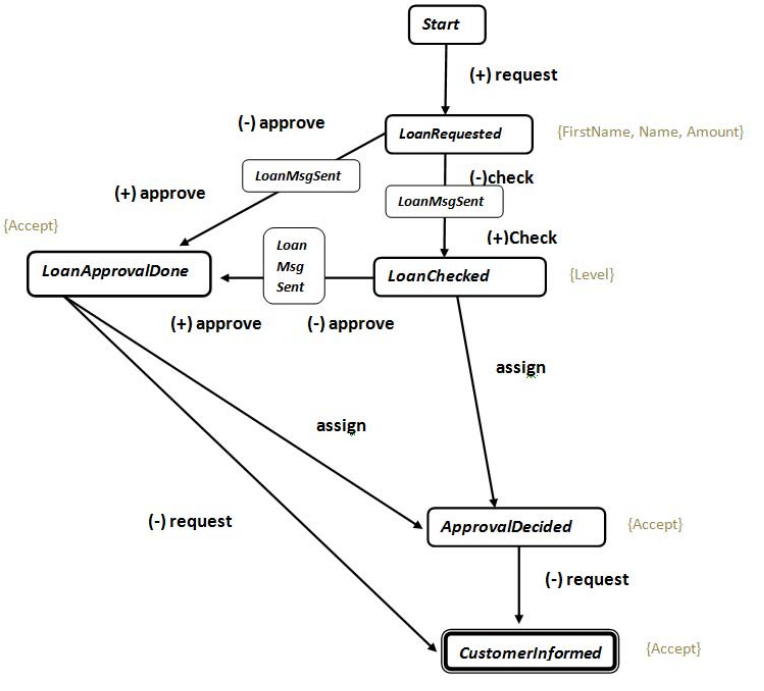

Fig. 2. The business protocol for the Loan approval example

role is to evaluate the risk of accepting the loan (expressed by the (-)Check message) or to check it directly by the approval service (expressed by the (-)Approve Message). If the risk is low, the acceptance will be decided locally and then assigned to a local variable that will be transmitted to the customer. Otherwise a processing from the approval service is needed and this latter has the responsibility to directly inform the customer in case of refusal or to return the response to the loan service that will forward it to the customer in case of acceptance. In both cases the customer is informed of the result of her/his request.

We first give some definitions that will serve introducing our monitoring language, then we will provide a syntax.

4.4.1 Execution Paths

As defined in the work [4] all traces left by the execution of a business process are captured by the corresponding business protocol. In the above example, the sequence **Start, request(+), LoanRequested, check(-), LoanMsgSent, check(+)LoanChecked** is an execution path. A complete execution path is an execution path that starts with the first state of the protocol and ends with its final state. It denotes a complete execution of the process represented by this protocol.

Definition 1. *Given a business protocol P, an execution path is formed by all the nodes (states of P) and edges (transitions of P) traversed during an execution. All the instances of the execution of one process generate execution paths that will be represented in a tree of executions. Figure 3 represents four paths of four different instances identified by their instance ID.*

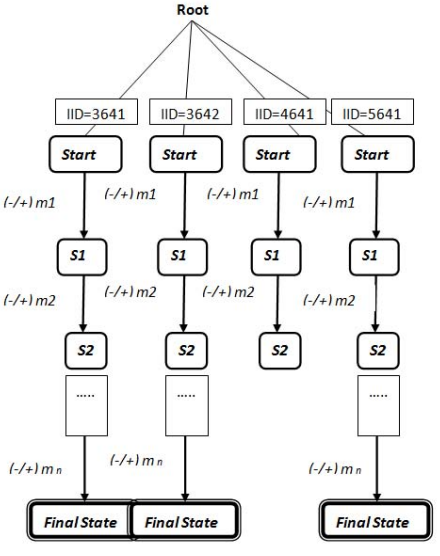

Fig. 3. A tree of all execution instances

Definition 2. *Let us consider a business protocol P defined as a tuple P=(Q, q_0, F, ϕ, Σ,ψ, Var). A query over a business protocol P is a function that takes a path expression as input, that is a start node, an end node and eventually a set intermediate nodes constrained with the names of states defined in P (that is Q). It returns the values of variables defined on those nodes, an aggregation of those variables or a number of paths.*

4.4.2 The Query Language Syntax

A query over the business protocol P is an expression built using the the syntax shown figure 4, where terminals constitute keywords of the language and non-terminals are used in production rules and are thus underlined.

A query is composed of three clauses:

- Retrieve
- Where
- Constrain

The **Retrieve** clause specifies the information that will constitute the answer. It could be an attribute or a set of attributes. It could also be an aggregate result on the number of selected paths or the average number of executions leading to the selected paths.

The **where** clause specifies the paths to select given a start node and an end node (the start and the end keyword respectively). We could restrict the selected paths by indicating intermediate nodes to cross or not to cross. The answer returned by the Retrieve clause is the set of attribute values of the selected paths if the query is intended to return attribute values, or an aggregate on these values or the number of paths that were selected. We can restrict even more the paths that will be selected using the **constrain** clause by fixing the values of attributes or the value of aggregates made on attributes values, or aggregates of time.

4.5 Query Evaluation

As mentioned previously, the queries formulated over the business protocol will be translated into SQL queries over an event database that captures the business process execution. First, we give the schema of such a database that will enable to retrieve the information as stated in the query language. Then, the above queries will be translated into their corresponding SQL queries over the database.

4.5.1 The Database Schema

The schema of the database is obtained by mapping each state of the business protocol to a relation of the database. Each relation is given the name of the state from which it is generated and the attributes identified in that state. Additional transformations are however required in case the variables defined in the business protocol (taken directly from BPEL specification which is XML-based) do not fit into relational table columns unless the host RDBMS allows storing such XML types.

Q → Retrieve Info where Path constrain constraints

Info → Projection / Special / state

Projection → State.Attribute , Projection / State.Attribute

Path → start = State NXT / start = State

NXT → end c1 State / end c1 State Foll / end c3 FinalStates

Foll → interm c1 State Foll/ interm c1 State

constraints → ctr AND ctr / ctr

State → S_0 ,.....S_n

Attribute → Att_1 .....Att_n

ctr → State.Attribute c1 atype / State.Attribute c2 ntype / Time C ntype/ Agg

Special → Agg / ctr / Time

Agg → average (Attribute) / sum (Attribute) / count (State)

Time → Time / average (Time)

C → c1 / c2

c1 → =/ ≠

c2 → ≤ / ≥ / < / >

c3 → ∈ / ∉

Fig. 4. The query language syntax

Each state is designated by the ID it will have at run-time which is given by the BPEL engine to every running instance. At a given time, each state is linked with one and only one state (the following state in the execution path). The resulting table from a given state has the Instance ID (IID) as primary key, the variables of the state as attributes and a 1 to 1 multiplicity with the states coming right after it in the protocol representation.

For simplicity, we consider the example of figure 5 that shows the database schema resulting from the transformation of the protocol of figure 2. The intermediate states of the protocol (states without variables) are not mapped to any table in this schema. They are, however, stored elsewhere in a table called

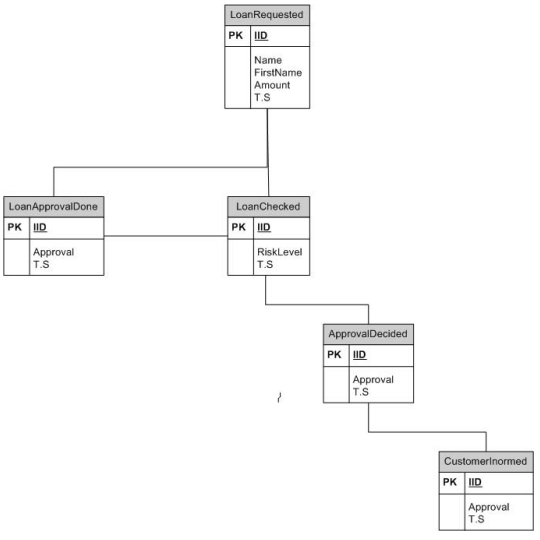

Fig. 5. A database schema of the protocol in figure 2

'Actual' that given an instance ID returns the name of the last state reached by the execution of an instance (the states in the business protocol). This table can be populated following two ways: each time a different instance is inserted into a table representing one state of the protocol, the 'state' attribute of the 'Actual' table is updated with the name of that table for the same IID. This is done by associating a trigger to every table. On each tuple insertion to a table, the name of this table is inserted into the state field of the 'Actual' table with teh schema:

`Actual (IID,state, status,timestamp)`

where: IID is the primary key and corresponds to the IID of the table where a tuple is inserted, state is the name of the table where the tuple is inserted, timestamp is the instance of time when the tuple was inserted into the original table and status has special significance which will be explained after. The trigger of a table resulting from a state S_i can be defined as:

```
CREATE TRIGGER state_i_run
ON INSERT ON state_i_table
DECLARE
--X will hold the IID of the inserted tuple
X
BEGIN
-- If a tuple with the same IID already exists
IF X IN (SELECT IID FROM Actual) THEN
-- Update only the 'status' field
UPDATE Actual SET state='state_i'
ELSE
```

```
-- if the instance has not been yet recorded
INSERT INTO Actual (X,state_i)
END
```

Another way to populated the table is done at the level of the business process enactment by capturing messages sent in an invoke activity (cf. §4.3.1) that have not yet been responded by the partner link (if a response is required). The information which will be stored is the name of the partner link involved. Without this information we would never be capable of tracking the processes involved in failure or estimate their response time. Indeed, this prevents from mapping the intermediate states that denote in the business protocol that a message request has been sent and a response is expected.

At run-time, each created instance of the business process is stored in the database by filling the suitable fields with information generated during the execution. Each row of the database table is timestamped to enable the retrieval of temporal information.

The duration of a complete execution path is then given as the difference between its final and initial states' timestamps.

5 Conclusion

In this work, we provided a preliminary framework for business process monitoring using queries. This is just a starting work that will be helpful in:

- providing an abstraction of the monitored process that captures enough details relevant to monitoring issues, and not too much that could hinder the understanding of the modeled process;
- allowing an intuitive query formulation by visually selecting and eliminating parts of the process abstraction;
- ensuring efficient query evaluation by relying on relational databases that turn out to be more useful than expected when exploiting related mechanisms such as statistical analysis, actions triggering using the ECA paradigm but also off-line analysis since data is made persistent.

This work suggests reconsidering the problem of monitoring by taking another look that may lead to a solution when a important number of requirements will be satisfied. This is why we consider extending the high level query language so that it can deal with the maximum of situations one could need when monitoring any kind of process. This could be done by defining another syntax or extending the actual one while ensuring semantically correct queries with regards to a convention that will be made. A semantic compilation has to be defined at this level of abstraction so that high level queries will be mapped to the right SQL ones.

Since this monitoring works jointly with the BPEL standard specification, a deep review of the abilities of this latter could be of great benefit for optimization issues. for example, we could exploit the exception handling mechanisms defined in BPEL rather than redefining another one.

Additional extensions may concern querying flow activities after representing them and providing the suitable transformation mechanisms.

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., Koenig, D., Marin, M., Mehta, V., Thatte, S., Rijn, D., Yendluri, P., Yiu, A.: Web services business process execution language version 2.0 (OASIS standard). WS-BPEL TC OASIS (2007), `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html`
2. Baresi, L., Di Nitto, E.: Test and Analysis of Web Services. Springer, Heidelberg (2007)
3. Benatallah, B., Casati, F., Ponge, J., Toumani, F.: On temporal abstractions of web service protocols. In: Belo, O., Eder, J., Cunha, J.F., Pastor, O. (eds.) CAiSE Short Paper Proceedings. CEUR Workshop Proceedings, vol. 161, CEUR-WS.org (2005)
4. Benatallah, B., Casati, F., Toumani, F.: Analysis and management of web service protocols. In: Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.W. (eds.) ER 2004. LNCS, vol. 3288, pp. 524–541. Springer, Heidelberg (2004)
5. Wombacher, A., Fankhauser, P., Neuhold, E.J.: Transforming BPEL into annotated deterministic finite state automata for service discovery. In: ICWS, pp. 316–323. IEEE Computer Society, Los Alamitos (2004)