Applying Protocol Service for the Monitoring of Business Process

Xiaofeng Du[1], Behzad Bordbar[1], Mohammed Alodib[1], and Basim Majeed[2]

[1]School of computer Science, University of Birmingham, UK
x.du@bham.ac.uk, {m.i.alodib, b.bordbar}@cs.bham.ac.uk
[2]Etisalat BT Innovation Centre (EBTIC), Khalifa University, Abu Dhabi UAE
basim.majeed@bt.com

ABSTRACT

Identifying occurrences of failure is a major challenge in business process management. Current business monitoring techniques are invaluable tool for debugging and evaluating business process executions. In this paper a method of extending existing monitoring capabilities of the systems to allow real-time or near-real-time monitoring is discussed. The presented method draws on the existing Model Driven Development techniques to automatically create a new module called Protocol Service. The created Protocol Service is deployed with the existing system to monitor occurrences of undesirable events and failure. In this paper the outline of the approach will be presented.

INTRODUCTION

A business process involves a series of activities to produce products or deliver services to customers. A large enterprise can have very complex business processes to deliver services to their end customers. To ensure services being delivered smoothly and correctly, an effective Business Process Management (BPM) system is crucial. A BPM system can not only help enterprise to govern their business processes, but also accelerate process improvement and facilitate process innovation [13]. A key requirement for the success of BPM is the availability of robust business process monitoring functionality for performance, compliance, and risks. Business process monitoring provides the tracking information of each process instance's state. The information can then be used to analyse the performance of the processes or problem diagnosis. As the market trends change fast and customers demand better and faster services, more capable process monitoring methods are required to help enterprise quickly identify problems and improve their services, especially when business processes are complex.

There are several process monitoring applications available on the market, such as Oracle BPEL Process Manager [9] and ProM [10]. However, there are still some problems that need to be improved or solved. Here we assume that processes are described using BPEL. The identified problem areas are as following.

- **Real-time or near real-time monitoring**. Due to fierce market competition, an enterprise needs to closely monitor their business processes, improve services to satisfy new market needs, and quickly identify and recover any process failures. The current process monitoring technologies are mostly based on system log, which means that when a failure is identified, it has already happened. To prevent delivering wrong or faulty services to customers, failure should be ideally identified and recovered before the execution of a process ends. In this case, a real-time or near real-time process monitoring technique is essential. Almost all the current BPEL execution engines have real-time BPEL execution monitoring functionality. However, these monitoring methods are mainly provided to BEPL developers for debugging purpose.
- **Limited monitoring functionality**. Most of current applications for business process monitoring are not open source and the monitoring functionality cannot be extended by end users. These applications either focus on analysing the data collected during process execution and producing reports after the execution is completed or focus on runtime exception handling and reporting. However, to comprehensively monitor a BPEL process, we need not only exception handling and reporting, but also other monitoring functionality, such as message content monitoring, service failure monitoring (but not represented as runtime exceptions), and Service Level Agreement (SLA) compliance monitoring. Currently, one process monitoring application hardly suites various monitoring requirements.
- **Reusability**. As discussed previously, to keep up with fast changing market trend, a business process must be often updated. The update can be either updating a business process to a more efficient one or updating an existing standard to a newer version. A process monitoring system must be able to cope with the changes of processes without requiring large amount engineering work.

The structure of the paper is organised as follows: we first give some background knowledge of BPEL and Model Driven Development; then we discuss the detail of how the Protocol Services are generated using MDD based method and how a BPEL process is monitored; finally, we review related work and conclude.

PRELIMINARIES

BPEL and BPEL Monitoring

The Business Process Execution Language for Web Services (BPEL) [3] provides an XML based-language for the formal specification of business processes and

business interaction protocols. A BPEL file makes use of the WSDL file of involving services. Consequently, BPEL can be seen as an extension of WSDL [4] that provides basic one-way or request-response mechanisms for the Web service inter-communication. BPEL is designed for expressing processes in detail, allowing composition and coordination of activities such as for sequential, parallel, iterative, conditional, compensational, and fault execution [15]. Hence, business process expressing interaction between services can be specified elegantly.

Almost all of the BPEL execution engines provide BPEL monitoring facilities, such as the Oracle BPEL engine (embedded in Oracle SOA Suit) [9] and the Sun BPEL engine (embedded in Open ESB) [8]. The BPEL monitoring facilities within these BPEL engines can give BPEL developers indication of where errors happen during a BPEL instance execution and stack trace for the errors.

MDD and Model Transformation

In this paper, we shall make use of Model Driven Development (MDD) [14] to automatically modify existing BPEL. Central to the MDD is the process of Model Transformation, which automatically generates a new model from an existing one. Fig. 1 depicts an outline of MDD and the process of Model Transformation. A number of Transformation Rules are used to define how various elements of one metamodel (source metamodel) are mapped into the elements of another metamodel (destination metamodel). The process of Model Transformation is carried out automatically via the software tools which are commonly referred to as Model Transformation Frameworks [6] [1].

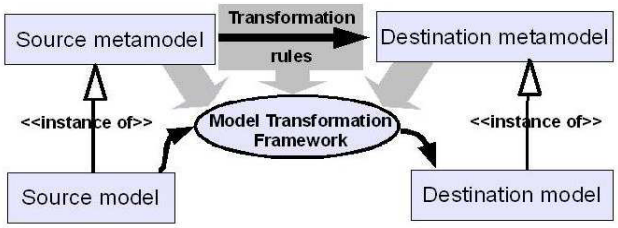

Fig. 1. Model Driven Development

A typical Model Transformation Framework requires three inputs: source metamodel, destination metamodel and transformation rules. For any instance of the source metamodel, the Model Transformation Framework executes the rules to create an instance of the destination metamodel. One way to express such rules is through Query/View/Transformation [5]. QVT is a standard for expressing MDD model transformations governed by the Object Management Group (OMG). Further reading on QVT could be found in [5].

SKETCH OF THE SOLUTION

In this paper, we propose a process monitoring method to tackle the problem areas discussed above. The method provides a near-real-time process monitoring mechanism by creating an embedded *Protocol Services* inside a BPEL process to generate a near real-time BPEL diagnoser, see Fig. 2. The diagram illustrates that a BPEL process is embedded with Protocol Service after the transformation tool. The original Partner Links (PL) of the BPEL process is replaced with the Protocol Service's PLs and the original PLs are managed by the Protocol Service.

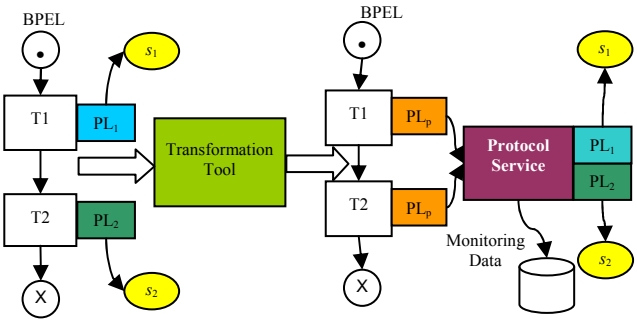

Fig. 2. Protocol Service generated and embedded into BPEL process.

The monitoring mechanism is based on Discrete Event System fault diagnosis theories [12] and therefore, it can notify not only where and when faults have happened, but also the types of faults. By embedding Protocol Services, all message traffics between a BPEL process and the target services in its Partner Links will be re-routed through Protocol Services. A user can extend Protocol Services to monitor the aspects of a BPEL process that he is interested in. The BPEL diagnosing service and Protocol Services are automatically generated using the model transformation technique SiTra [1]. It can easily cope with changes in BPEL process without large amount re-engineering work.

PROTOCOL SERVICE GENERATION

The Protocol Service acts as a proxy between the BPEL and the target services in the original PLs of the BPEL. As shown in Fig. 2, when an original BPEL process passes through the transformation tool, its original PLs are replaced by the PLs of the Protocol Service. The original Partner Links of the BPEL process are restored inside the Protocol Service. When the BPEL process is executed, it only talks to the Protocol Service and the Protocol Service can figure out what the target service is in each service request and contact the target service to get the result back to the BPEL process.

The transformation tool is a MDD based tool and uses the model transformation method to generate the modified BPEL with Protocol Service. The detail of the transformation method is described as follows.

Sketch of the Transformation

The transformation method requires three items to perform the mapping: the metamodel of BPEL as source, the metamodel of BPEL as destination, and the

transformation rules from the original BPEL to the BPEL with Protocol Service integrated. Due to space restriction, we could not include the metamodel here, but we refer the reader to [2] for more information. The transformation rules can be specified as follows. Firstly, the tool produces a BPEL service with conditional statements for the Protocol Service. This conditional statement is represented by using a Switch activity with multiple Cases. Then, the tool parses each Service individually to find any Invoke activities. Each Invoke activity is transformed to another Invoke activity by replacing its original PL to the WSDL file of the Protocol Service. The name of the original PL should be passed as an input to Protocol Service. Then, a Case will be added to the Switch activity in the Protocol Service. This Case has an Invoke activity which its PL is assigned to the original PL that was replaced in the previous step.

The generated Protocol Service is a skeleton with some basic implementation. Because of its abstract design, users can extend its functionality to monitor their required aspects of business process. As the Protocol Service has the full visibility of the content of the messages exchanged between the BPEL process and the services involved in the process, very detailed process monitors can be created to help in finding out why a failure happens, e.g. invalid inputs or internal system faults, where exactly the failure has occurred, and when. By having the detailed information about process failure, a process designer can create strategic solution to prevent the same failure happening again in the future. For example, if human errors are the main reason for a type of failure, then additional training can be introduced. If the failure is caused by poor design, then process designers can consider redesigning the problematic process. If it is the limitation of the hardware or software capability, then upgrades and allocation of additional resources can be arranged.

BPEL PROCESS MONITORING

Once the Protocol Services are embedded into BPEL processes, they can act as sensors in the system for fault diagnosis. In other words, after the transformation tool the BPEL has been annotated by identifying observable events (those with Protocol Service PLs) and unobservable events. Then, a model transformation (BPEL2FSM) will be used to transform the annotated BPEL models automatically to a Deterministic Automaton so that the classical theories of diagnosability [12] can be applied to compute and create a Diagnoser.

We have implemented the presented approach as a Plugin for Oracle JDevloper. The implementation is an extension of our pervious approach [2] and follows the outline of the method as depicted in Fig. 3. The tool requires passing all annotated BPEL files and their XML Schema Definition (XSD) as inputs. Each BPEL file and its XSD are combined together to collect all required details, to transform the BPEL representation into the equivalent Deterministic Automaton. The first transformation (BPEL2FSM) is implemented via SiTra.

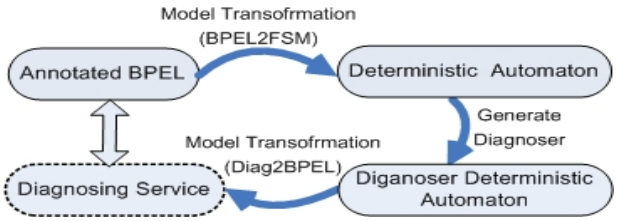

Fig. 3. Applying MDA to the design of Diagnoser

To create the Diagnoser Automaton the diagnosability algorithms are applied, e.g. as implemented in GIDDES and UMDES-LIB [11]. In our implementation, UMDES-LIB is used, which creates a Diagnoser Automaton from a given Deterministic Automaton. Finally, the created Diagnoser Automaton is transformed into a BPEL representation by using the second transformation method (Diag2BPEL) which is also implemented via SiTra.

The produced Diagnoser will be integrated into the system with a new service called the Protocol Service which is used to control all interactions between existing services and the created Diagnosing Service. All interacting services should send their request to the Protocol Service performing the invocation of the destination service which returns the invocation result and its current state to the Protocol Service. To determine the system behaviour after the invocation, the Protocol Service interacts with Diagnosing Service to perform the diagnosing tasks. Then, the diagnosing result will be provided to the Protocol Service which returns this result with the invocation results received from the destination service to the invoker service (source service). Fig. 4 shows an example of a business process which has two Invoke activities. It can be seen that the interaction should be carried out through the Protocol Service.

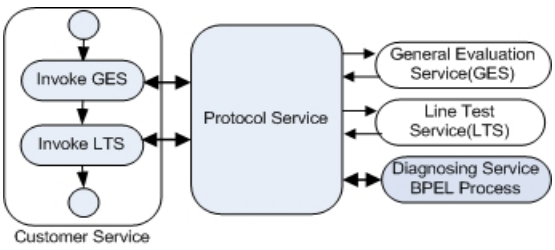

Fig. 4. Applying MDA to the design of Diagnoser

As the diagnoser monitors the state of a BPEL process and identifies failure in real-time, it can provide valuable information for failure recovery at runtime before the process delivers the wrong products or services to the end customers. The type of the failure can also be identified at the same time so that the failure recovery process can be more effective and efficient.

The Protocol Services embedded in the BPEL process can act not only as sensors, but also as history data recorder. As shown in Fig. 1, the Protocol Service can record event data, such as service failure, unavailability, and response latency, into database. This kind of information can help to predict a service's reliability during a process design phase. For example, a process designer may not be aware of recent problems

experienced by some of the services that are going to be used in the new process. In this case, the history data about these services can be retrieved from the database and represented to the process designer with indication of predicted future behaviour of the services as a reference. If the data shows that a service behaved in an unstable manner and its failure rate in the past month is 5 failures out of 10 invocations for example, the process designer may need to seek alternative services.

RELATED WORK

In [7], Baresi and Guinea provide a proxy-based solution to support the execution of BPEL monitoring rules at runtime. It dynamically integrates monitoring rules into the process through a method. In [16], Yan and Dague propose a Model-Based approach to diagnosing of behaviour of Web services by extracting synchronized automata from the BPEL. Our approach differs from the above work in various ways. Firstly, we make use of MDD to automatically generate the protocol services and the BPEL diagnoser. Secondly, using MDD allows us to reuse existing results in DES [12] and UMDES tool [11] which reduce the cost of implementation. Finally, our approach fundamentally differs from the above as our diagnoser is modelled in Web services languages.

CONCLUSION

In order to survive on the highly competitive market, enterprises must be able to closely monitor their business processes, quickly steer their processes to suite new customer needs, and identify and recover from failures immediately to minimum service disruption. However, as discussed in this paper, there are several problem areas that need to be addressed to produce an efficient and effective process monitoring method.

In this paper, we proposed a business process monitoring method that solved or improved the problem areas discussed previously. First of all, it is a near real-time business process monitoring method. It uses the DES fault diagnosis methods to identify faults in near real-time; secondly, its monitoring ability is extendable upon user requirements; and finally, the model transformation technique is used to generate Protocol Services and BPEL diagnosing service and it makes the monitoring method highly adaptable to changes without large amount of engineering work.

REFERENCES

[1] Akehurst, D.H., Boardbar, B., Evans, M., Howells, W.G.J., and McDonald-Maier, K.D. (2006) SiTra: Simple Transformations in Java , ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems, LNCS, Vol. 4199, pages 351-364, 2006

[2] Alodib, M., Bordbar, B., Majeed, B. (2008), A model driven approach to the design and implementing of fault tolerant Service oriented Architectures, IEEE International Conference on Digital Information Management , London, UK: pp 464-469

[3] Business Process Execution Language for Web Services, Version 2.0. (2006), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html

[4] Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S. (2006) Web Services Description Language (WSDL) Version 2.0, W3C (2006), http://www.w3.org/TR/wsdl20/

[5] Gardner, T., Griffin, C., Koehler, J., Hauser, R. (2002) A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard. OMG, ad/03-08-02 (2002)

[6] Jouault, F. and Kurtev, I. (2006) On the Architectural Alignment of ATL and QVT, In Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 06). ACM Press, Dijon, France, chapter Model transformation (MT 2006), pages 1188--1195

[7] L. Baresi and S. Guinea. (2005) Towards Dynamic Monitoring of WSBPEL Processes. In 3rd International Conference on Service-Oriented Computing (ICSOC'05), pages 269–282, 2005

[8] Open ESB (2010) https://open-esb.dev.java.net/

[9] Oracle SOA Suit (2010) http://www.oracle.com/technology/products/soa/soasuite/index.html

[10] ProM (2009) Process Mining Tool Kit, http://prom.win.tue.nl/tools/prom/

[11] Ricker, L., Lafortune, S. and Genc, S. (2006) DESUMA: A Tool Integrating GIDDES and UMDES. in 8th International Workshop on Discrete-Event Systems. 2006.

[12] Sampath, M., Sengupta, R. and Lafortune, S. (1995) Diagnosability of discrete-event systems. IEEE Transactions on Automatic Control, Sept. 1995. 40: p. 1555-75.

[13] Scheible M. (2007) Exploring key facts about business process management with IBM WebSphere software, Business process management solutions White paper, IBM, February 2007

[14] Stahl, T. and Volter, M. (2006) Model Driven Software Development; technology engineering management: Wiley, 2006.

[15] Web Service Choreography Interface (WSCI) 1.0, W3C Note (2002)

[16] Yan, Y. and Dague, P. (2007) Modelling and Diagnosing Orchestrated Web Service Processes, in In Proceedings of the ICWS 2007.