Towards a Unified Framework for the Monitoring and Recovery of BPEL Processes

Luciano Baresi, Sam Guinea, and Liliana Pasquale
Politecnico di Milano
Dipartimento di Elettronica e Informazione
Via Golgi 40, 20133 Milano
{baresi | guinea | pasquale}@elet.polimi.it

ABSTRACT

Web services have proven to be a viable solution for interoperability issues. Since end users do not buy services, but only interact with them remotely, such complex systems end up having a distributed ownership, meaning different parts of a system can evolve independently. This has brought researchers to concentrate on run-time management issues such as dynamic monitoring and self-recovery.

However, we advocate that no silver bullet has been found. All the major approaches have advantages and disadvantages. In this paper we propose a unified framework for monitoring and recovery that provides a clear separation between data collection and analysis, a common management infrastructure, and a common recovery system. Separating monitoring from recovery allows the framework to integrate different monitoring approaches seamlessly through a plug-in approach. The common management infrastructure allows us to dynamically manage the multiple monitoring approaches being used, while the common recovery approach allows us to activate advanced recovery techniques both on process instances and process definitions.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program verification—Assertion checkers, Reliability; D.2.5 [**Software Engineering**]: Testing and Debugging—Monitors, Error handling and recovery

General Terms

Design, Verification, Management

Keywords

web services, BPEL, quality of service, monitoring, recovery

1. INTRODUCTION

Service Oriented Architectures are a proven architectural style for the creation and execution of large and complex distributed systems. Among the various existing service technologies, Web services happen to have a very florid research community. They can be seen as black-box components that are deployed on the web, and accessed using standard web protocols. However, they have a very distinct peculiarity: they remain under the service provider's jurisdiction at all time. Clients do not acquire the component, they just interact with it remotely. Although it is just a small technical difference, this change has profound consequences on how a system is designed and managed.

First of all, design mainly consists in choosing and composing services that are provided remotely by third parties. The BPEL standard (Business Process Execution Language) [2] is a good example, as it has imposed itself as the de-facto way of composing services into orchestrated processes. The delegation of key aspects of a system to a third party means that we need to believe it will contribute to the system's overall functionality and quality of service (QoS). Unfortunately, since remote services are free to evolve independently of the main application, and since design by contract is not a common practice amongst service developers (providers seldom give more than a syntactical specification of a service's interface), even a modification of the service, considered to be an improvement by the provider, can be harmful to the system. Second, although services have often been intended as merely a means to solve interoperability issues in business to business scenarios, they have also begun to emerge as a valid approach for novel computing models, such as pervasive computing, ambient-intelligence, and context-aware computing [8]. In these models it is important to be able to modify a system's behavior at run time, taking decisions based on the situation at hand. This is facilitated by dynamic binding techniques. For example, BPEL allows the definition of abstract processes, and allows the actual service endpoints to be discovered at run time. The result is that, at design time, it is practically impossible to have a complete vision of the system, and of the configurations it may assume at run time. Classical validation steps, such as static verification and testing, are no longer sufficient, and demand that we consider run-time management. Indeed, in the last few years research has rightfully concentrated on solutions for guaranteeing QoS in such dynamic scenarios. In particular, run-time monitoring of service compositions has been the focus of much discussion.

The goal of monitoring a service composition is to become aware of any erroneous behaviors the system may have dur-

TAV-WEB – Workshop on Testing, Analysis and Verification of Web Software, July 21, 2008

ing execution. It is beneficial since we do not need to wait for the system to crash before trying to adapt its behavior. Many approaches have been proposed, with different advantages and disadvantages. They mainly differ in the requirements they deem important. For example, some claim that monitoring should be a non-intrusive operation [12], as to not disrupt the execution, while others profess that monitoring can be intrusive if this allows us to be more timely in discovering failures [6]. In this paper we advocate that no silver bullet for monitoring and recovery has been achieved. The advantages and disadvantages of the different approaches are undeniable, meaning that none is perfect and that none should be entirely ruled out. Given our extensive experience in the field [6, 4, 7], we propose to address the problem by means of a unified monitoring and recovery framework for BPEL processes. In this paper, we will give a snapshot of our ongoing work, stressing the overall picture and underlining the main open issues that remain to be conquered.

Our vision for a unified framework builds upon three main pillars. First of all, the framework should decouple data collection from data analysis. In fact, a uniform data collection model would allow the integration of many different analysis techniques, allowing us to exploit their main benefits. Second, the framework should provide a common management infrastructure, for organizing how the different approaches fit together, and for tailoring how much monitoring is being achieved. Third, the framework should also provide a common infrastructure for the recovery of the BPEL processes. Our goal is to encourage advanced research in self-recovery techniques by offering a testbed framework.

We believe that the framework will allow us to integrate our previous efforts in this fields, and to evaluate the benefits of having complex monitoring strategies that mix different approaches. The rest of the paper is organized as follows. Section 2 presents both the authors' existing body of work and the current state of the art in monitoring and recovery of service based systems. Section 3 introduces the unified framework, while Section 4 concludes the paper by giving a clear picture of the work that has been achieved and the work that still remains to be completed.

2. STATE OF THE ART

Current BPEL engine implementations, such as ActiveBPEL [1] are inherently static and they only support primitive forms of probing and exception handling. Probing capabilities are limited to internal variables and timeouts, and any reaction to exceptional behaviors must be intertwined and deployed with the main execution flow. There is no way to customize the reaction of the system with respect to context information and user requests.

To overcome these limitations different monitoring and recovery techniques were proposed. Monitoring approaches can be classified in several different ways: timeliness, invasiveness, nature of verifiable properties, etc. Our goal here is not to give a complete and precise presentation of all the existing monitoring and recovery approaches, but to give a flavor for what researches have accomplished, and how none of them provides an exhaustive solution. Due to space constraints we have chosen to dedicate slightly more space to the authors' body of previous work. The choice was made to give the reader the minimum amount of detail needed to appreciate the rationale of the rest of the paper.

The authors proposed the Dynamo framework [6, 5] for the monitoring of BPEL processes. This work assumes that designers not rely on providers giving formal specifications of their services, but adopt a defensive approach to designing their processes. Using a specially defined constraint language (WSCoL), designers are asked to define punctual assertions (pre- and post-conditions) on the interactions the process has with its partner services. These assertions are maintained separate from the business logic, and Dynamo used AOP (Aspect Oriented Programming) [11] techniques to weave the verification activities into the execution at run time. In practice, Dynamo uses a synchronous approach in which the process is temporarily stopped while the assertion is verified. Although quite an invasive approach, and potentially detrimental for performance, anomalies are detected as soon as they occur, leaving the system in the most appropriate state to perform self-recovery.

The authors continued their work with the introduction of a completely new approach based on the ALBERT (Assertion Language for BpEl inteRacTions) [4]. Using this assertion language, designers are required to define process invariants. Once again, the assertions are kept separate from the business logic, and AOP techniques are used to bridge the process' normal execution and the monitoring activities. The main difference with respect to WSCoL is that ALBERT adds temporal predicates and temporal functions. In this paper we do not focus on how ALBERT works and handles the verification of temporal logic properties, but interested readers can find an in-depth example in [4]. Since temporal properties are no longer punctual, they cannot be checked in a single state and they need to be verified over time. As a consequence, the process is only stopped momentarily to gather information, effectively minimizing the degree of invasiveness. The run-time verification, on the other hand, is achieved in a parallel thread. The main disadvantage is the anomalies are detected when the process is already beyond the state in which it might make sense to perform recovery. Rollback techniques are being investigated, but this is a research topic in itself.

If we look at approaches that have proposed by others, it makes sense for many of them to be integrated in a unified framework such as the one we are building. For example, Erradi et al. [10] propose to analyze how a system behaves with respect to the intersection of client-side and provider-side policies expressed using WS-Policy. Mahbub et al. [12], on the other hand, collect system execution events, place them in a persistent storage, and perform off-line analysis based on a form of event calculus.

The authors have also tackled the run-time recovery of BPEL processes [7]. Using a special recovery definition language (WSReL), designers can pick from a set of pre-defined atomic recovery actions to combine them and build more complex recovery strategies. In the approach, all atomic actions have process instance validity, meaning no recovery is performed on the process definition itself.

Recovery has received, in general, less attention from the research community. For example, we are still far from having adequate solutions for issues such as dynamic instance/class re-configuration or process rollback. On the other hand, there has been work in the field of dynamic rebinding. For example, Ardagna et al. [3] propose the PAWS (Processes and Adaptive Web Services) framework, in which services are dynamically chosen in order to optimize given

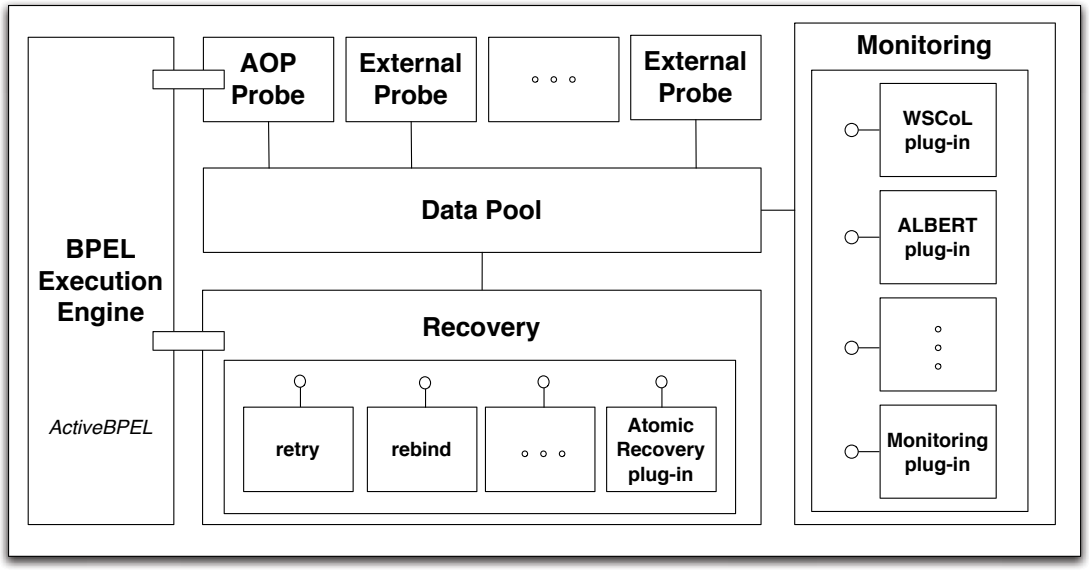

Figure 1: The Unified Framework.

QoS levels. Colombo et al. [9], on the other hand, choose their partner services depending on SLA definitions. They use an ECA (Event-Condition-Action) rule system to discover when an SLA infringement occurs, and to switch to a service that can allow the system to proceed in its normal execution.

3. TOWARDS A UNIFIED FRAMEWORK

A unified supervision framework (i.e., monitoring and recovery) for BPEL processes requires a common data collection model, a common management infrastructure, and a common recovery framework. On the other hand, data analysis techniques can be added in a pluggable fashion. A tentative high-level run-time architecture is given in Figure 1. It reflects the conceptual separation of concerns between data collection, data analysis, and recovery.

Data collection exploits two kinds of data probes. The first kind is `AOP probes`. These probes are directly connected to the process' execution environment using AOP techniques. They briefly stop a running process to gather its internal state of execution. The only downside is that to produce the AOP probes we need to have access to the BPEL engine's source code. (Our current implementation uses the latest version of ActiveBPEL's engine.) The second kind is `external probes`. These probes can be conveniently placed throughout the environment to gather run-time context information. The advantage of such probes is they can be used regardless of the execution environment chosen. For example, they can be placed on the engine's transport channels to gain information about the data it is exchanging with its partners, or they can be used to listen to an engine's run-time events. The difference with AOP probes is that they cannot be used to capture internal data changes, for example due to BPEL Assign activities.

Once the data has been collected, they are time-stamped and placed in the `Data Pool`. This is currently implemented using tuple space technology, allowing us to separate data collection from the other two main activities: data analysis and process recovery.

The `Monitoring` component is configured to react to the presence of certain data within the data pool. When the data appears, they are passed to the appropriate monitoring plugins which proceed to perform analysis. Completed the analysis, the results are placed in the data pool for three reasons. The first is to resume a blocked process. This occurs when the nature of the analysis is synchronous (e.g. a pre- or post-condition on a BPEL Invoke activity). The second is to share partial analysis results between montioring plugins, while the third is to activate recovery.

Finally, the environment provides a uniform `Recovery` system, based upon pluggable atomic recovery caabilities that have access to the process execution using AOP hooks. This allows them to change the way the process will play out from that point on.

We will now discuss the three main pillars upon which the unified framework stands: the separation of concerns between data collection and data analysis, a common management infrastructure, and a common recovery framework.

3.1 Data Collection vs. Data Analysis

To have a complete view of the system, we are interested in data extracted from the process' state, from the environment in which it is being run, and from historical traces of previous executions. In our current implementation we distinguish between two kinds of data (internal and external). For the former, designers must indicate what data they want as well as the *location* in the process in which it is to be collected. AOP probes are then used to stop the process at the defined location, and to gather the required data. For the latter we gather data from sources that are external to the process execution, i.e., from probes that are conveniently placed for monitoring purposes. This is needed when the correctness of a system depends on the context in which it is being run. Regarding external data we allow two kinds of collection: sychnronous and asynchronous. In the first

case, a designer must indicate the location in the process in which the data needs to be collected. This is necessary when the external data we collect depends on data coming from within the process. For example, an external data collector could be used to transform internal data depending on context information. In the second case, the external probe is configured to periodically collect the data and to place them in the data pool.

On top of defining how and when data needs to be collected, designers must specify persistance policies for the data that is added to the data pool. This allows us to avoid an uncontrolled growth of the pool, as well as to define the historical traces needed by monitoring apporaches that predicate on data sequences, such as Albert. Current policies can be `single instance`, in which the pool overwrites previous data when a new instance arrives, `time-frame`, in which the pool maintains a single data for a given amount of time, or `space-frame`, in which the pool maintains a single data up to a threshold amount of new data instance arrivals. More policies can be added subsequently.

Thanks to clear and powerful data collection mechanisms, and to a plug-in-based architecture, the unified framework can exploit many different monitoring approaches. Notice that one of the advantages of having a clear separation of concerns between data collection and data analysis is that it allows us to share data collected from various processes, and to define cross-process analysis. Another advantage is that it allows us to optimze the data collection step, since data placed in the pool can be shared by more than one data analysis technique. As previously stated, we mainly distinguish between synchronous and asynchronous data analysis. The synchronous case is quite simple, and we have already prototyped it with our WSCoL plug-in. In this case, the process simply waits for the punctual assertion to be checked, and for the results to appear in the data pool. The asynchronous case, however, is more difficult. In this case we are interested in simply forwarding the data to the monitoring plug-in, and in allowing the process to continue its normal execution. (We have prototyped this with the ALBERT plug-in, and it could easily be done with other approaches as well, such as Mahbub et al.'s.)

3.2 Management

The advantages of having more than one monitoring approach running at the same time lay in how these can be dynamically controlled through a unified management infrastructure. Our goal is to have a means of dynamically modifying the amount and the nature of the monitoring activities being performed. The reasons can be manifold: a particular moment in the process' life-cycle, the context in which the process is being run, etc. For example, during a process' lifetime we might decide to start with an invasive approach to discover anomalies in a very timely fashion, and after a while switch some activities over to an asynchronous approach to favor performance. On the other hand, should the asynchronous approach capture an anomaly we could always switch back to a more punctual and timely synchronous approach.

The way we are currently achieving this is to associate each monitoring activity with meta-level information that can be used at run time to determine whether it should be performed or not. For example, each activity can be given a priority level, while a run-time threshold value could be used to separate those that need to be executed from those that don't. Other meta-level information could indicate that a monitoring activity needs to be performed with a given frequency, or within a given time frame. All these meta-level information are made accessible through the monitoring component, to guarantee that they can be modified after the process has been deployed. This allows these values to be changed as a reaction to an anomalous behavior. We are also investigating a high level management language for defining complex monitoring strategies. This will allow us to more easily combine completely different monitoring techniques, such as statistical analysis or post-mortem analysis, with a synchronous approach such as the one used in Dynamo. Hopefully, the unified framework will provide just the testbed we need to understand the intricacies and the advantages of having a complex interplay of different kinds of monitoring techniques.

3.3 Recovery

Finally, the third pillar is a common infrastructure for process recovery. Once again the use of a plug-in architecture is beneficial, since it allows the infrastructure to be extensible with respect to its actual capabilities. As we have already stated, not much advanced research has been achieved in terms of recovery. Therefore, one of our goals is to provide an ideal testbed for fostering new ideas. The advantages of providing a well-established AOP mechanism for interacting with the executing process are evident.

We will allow designers to add new atomic recovery actions, to experiment with recovery techniques. In particular, we are interested in exploring techniques that operate not on the process instance itself, but on the process class. For example, planning techniques could be used to evolve the process definition, to allow it to cope with situations that were not even contemplated during the process' design phase. Obviously, the recovery framework will also have special atomic actions for accessing the management infrastructure. This will allow the overall degree of monitoring, both in terms of activities being performed and approaches being used, to be automatically set depending on how the process is doing.

The use of various monitoring techniques can cause several problems, since they can make different, complementary and probably conflicting, decisions. We need to establish clear rules for deciding when and which recovery strategies are activated. A preliminary solution is to activate recovery strategies at different times, depending on the nature of the monitorings that caused them. Recovery due to sychronous monitorings are activated after the BPEL activity the process is blocked at, but before the process resumes. On the other hand, recovery due to asynchronous monitorings are achieved as soon as possible, but always immediately before a new BPEL activity is started. The other open issue is what happens when two or more recovery strategies need to be activated. If they are two synchronous approaches, the designer knows about them and can define a rule that the system itself can use to choose which one to execute, for example depending on context information that is only available at run time. This continues in the direction of our previous work in the recovery of BPEL. processes [7]. Asynchronous recoveries pose a slightly more complex problem. In fact, the designer does not know when and which recov-

eries will request activation at the same time. We solve this problem by asking the designer to specify a priority for each recovery strategy. The designer is also called to define how the system should interpret these priorities. For example, if the designer decides to use an *exclusive* strategy, only the recovery with the highest priority level will be executed. If the designer decides to go with an *all* strategy, all recoveries are executed and the order is given by their priority values. (Recovery strategies with the same priority level can execute in any order.) If the designer decides to not provide priority levels, the recovery strategies are executed in any order. Finally, we also allow the designer to specify a final location for each recovery strategy. If the recovery strategy has not been executed by then, it is canceled. As usual, we provide a flexible solution and the correctness of the recovery executions is up to the designer.

Other directions however will be investigated. Our goal, in fact, will be to provide a general means that can be used across the framework, regardless of the monitoring approaches being used. However, in some cases re-synchronization might not even be an issue. For example, should the monitoring approach be particularly time-consuming, instance recovery might be entirely out of the question. In this case, the monitoring results could be used for process class adaptation.

4. CONCLUSION

The continuous emergence of Service Oriented Architectures, and in particular of Web services, has pushed researchers' attention towards techniques for guaranteeing quality of service at run time. In particular, much research has concentrated on composite BPEL processes. Even though many intelligent solutions have been proposed, we advocate that no silver bullet has been found. This is why we present our ongoing work on a unified framework for the monitoring and recovery of such processes. Thanks to a clear separation of concerns between data collection and data analysis, the framework is able to accomodate many different kinds of monitoring approaches that can be added as plug-ins. There is no need to choose one over another, but we can exploit the benefits of multiple approaches. The framework also presents a common management infrastructure, which can be used to specify how the different approaches should be coordinated depending on the process' life-cycle and/or its context of execution. Finally, the framework also provides an extensible recovery infrastructure. We believe that the overall framework is an important contribution, since it will also prove beneficial in providing a concrete testbed for new monitoring and recovery ideas.

5. REFERENCES

[1] Activebpel engine architecture, 2006.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services, Version 1.1. BPEL4WS specification, May 2003.

[3] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A Framework for Executing Adaptive Web-Service Processes. *Software, IEEE*, 24(6):39–46, 2007.

[4] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of Web Service Compositions. *Software, IET*, 2007.

[5] L. Baresi and S. Guinea. Dynamo: Dynamic monitoring of ws-bpel processes. In *Service Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005*, volume 3826 of *Lecture Notes in Computer Science*, pages 478–483. Springer, 2005.

[6] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC 2005: 3rd International Conference on Service Oriented Computing*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2005.

[7] L. Baresi and S. Guinea. A Dynamic and Reactive Approach to the Supervision of BPEL Processes. In *1st India Software Engineering Conference*, 2008.

[8] Luciano Baresi, Elisabetta Di Nitto, and Carlo Ghezzi. Towards open-world software: Issue and challenges. In *SEW*, pages 249–252. IEEE Computer Society, 2006.

[9] Massimiliano Colombo, Elisabetta Di Nitto, and Marco Mauri. Scene: A service composition execution environment supporting dynamic changes disciplined through rules. In Asit Dan and Winfried Lamersdorf, editors, *ICSOC*, volume 4294 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2006.

[10] A. Erradi, P. Maheshwari, and V. Tosic. Ws-policy based monitoring of composite web services. In *ECOWS '07: Proceedings of the Fifth European Conference on Web Services*, pages 99–108, Washington, DC, USA, 2007. IEEE Computer Society.

[11] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.M. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 - Object-Oriented Programming, 11th European Conference, Proceedings*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[12] Khaled Mahbub and George Spanoudakis. A framework for requirements monitoring of service based systems. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 84–93, New York, NY, USA, 2004. ACM Press.